Un outil générique de conception et de révision coopérative de Bases de Connaissances s'appuyant sur la notion de situation

Luc Poittevin

Orsay
N° d'ordre :

Université Paris-Sud
U. F. R. scientifique d'Orsay

THÈSE

présentée pour obtenir
le grade de Docteur en Sciences
de l'Université Paris XI d'Orsay

Spécialité : INFORMATIQUE

par Luc POITTEVIN

Sujet :

Un outil générique de conception et de révision coopérative de Bases de Connaissances s'appuyant sur la notion de situation

Soutenue le 11 septembre 1998 devant la Commission d'examen composée de :

M.	Joost BREUKER	Rapporteur
M.	Alain GIBOIN	Examinateur
Mme	Marie-Christine HATON	Rapporteur
M.	Yves KODRATOFF	Directeur
Mme	Claire NÉDELLEC	Responsable scientifique
M.	Gérard SABAH	Président
M.	Jacques VIALLAT	Examinateur

Remerciements

Merci à Claire Nédellec qui a encadré ce travail de thèse. Toujours disponible et avisée, Claire a accepté, malgré tous les écueils menaçants, de superviser un jeune thésard déterminé à poursuivre un travail qui se risquait hors des sentiers balisés. Elle a su orienter et canaliser mon travail tout en me donnant la liberté et la confiance nécessaires au développement des idées exposées dans cette thèse. Nos nombreuses discussions ont largement participé à la formulation de ces idées. Elle m'a appris l'exigence de la recherche scientifique de haut niveau. Sa compétence, ses conseils, ses relectures et ses encouragements ont joué un rôle fondamental tout au long de ce travail.

Merci à Yves Kodratoff de m'avoir fait confiance et accueilli au sein de l'équipe Inférence et Apprentissage du LRI. Yves a été un directeur de thèse disponible et éclairé. Sous sa direction, l'équipe Inférence et Apprentissage est une école “de rigueur et de créativité”. Il est d'ailleurs remarquable que cette expression, empruntée à Marta Franova, s'applique aussi bien aux programmes informatiques développés par l'équipe, qu'au fonctionnement même de l'équipe.

Merci à Marie-Christine Haton d'avoir accepté de prendre le temps d'être rapporteur de ma thèse et de s'être déplacée depuis Nancy malgré ses nombreuses charges. Sa relecture attentive de mon manuscrit m'a permis d'y apporter quelques corrections et enrichissements notables (REVINOS ne pouvait m'aider pour faire ces révisions...).

Merci à Joost Breuker, de l'Université d'Amsterdam, d'avoir très gentiment accepté d'être rapporteur de ma thèse et d'avoir fait l'effort de lire mon manuscrit en français. Les discussions avec un grand spécialiste du domaine de l'acquisition des connaissances, co-auteur de l'approche KADS, ont été stimulantes.

Merci à Gérard Sabah de me faire l'honneur d'avoir accepté de présider mon jury de thèse. Ses travaux dans le domaine de l'intelligence artificielle et des sciences cognitives sont des sources d'émulation.

Merci à Alain Giboin pour l'intérêt qu'il porte à mon travail depuis le commencement. Je suis heureux qu'il ait accepté de faire le voyage depuis Sophia-Antipolis pour être membre de mon jury. Alain Giboin est le représentant de ces communautés très précieuses que sont les groupes de travail du PRC-IA, comme l'ex-groupe Explication. Ces groupes sont des lieux d'échanges et de discussions trans-disciplinaires qui donnent une réalité au terme “sciences cognitives”.

Merci à Jacques Viallat, sous-directeur des Caisses de Prévoyance et de Retraite de la S.N.C.F, d'avoir accepté de faire partie du jury, et d'être venu spécialement de Marseille pour cela. Ce travail doit énormément à M. Viallat qui a été immédiatement intéressé par ce projet de thèse CIFRE et m'a donné tous les moyens nécessaires tout au long de ces trois années passées aux C.P.R. à Marseille.

Merci à toute l'équipe Inférence & Apprentissage du LRI pour l'ensemble des discussions, conseils et critiques qui ont permis d'améliorer la qualité de ce travail, mais aussi pour la très bonne ambiance qui règne dans ce groupe. Je voudrais remercier en particulier Céline et Antoine, mais aussi Lola, Karine, Hervé, David, Fabien, Sébastien, Caroline, Michèle, Erick, et tous les autres... Je me dois aussi de remercier l'équipe technique et l'ensemble du secrétariat du LRI pour leur aide.

Merci à tous mes collègues des C.P.R. pour leur contribution à ce travail et pour leur sympathie. Merci à André d'abord, responsable des serveurs télématiques des C.P.R., pour sa gentillesse et son soutien technique. Merci à Christian, Jean-Marc, Robert, Marie-Hélène, Bernadette, Chantal, Solange, Jean-Pierre, Christophe, Roger, Yves, et tous les autres... Un merci particulier à madame Usai, à messieurs Jordan, Flambeau, Maury, Dussol, Madrigal et Marty.

Merci enfin à mes parents pour leur soutien et leurs encouragements.

Et merci à toi, Florence, d'avoir accepté toutes les contraintes que ce travail a imposées, et de m'avoir toujours soutenu. Je te dédie ce mémoire, à toi et à notre petite Lise.

Plan

"Je ne parviens pas à énoncer assez suffisamment de force ma conviction que la préoccupation de la Cohérence, si valable en Logique Mathématique, a été incroyablement destructive pour ceux qui travaillent sur les modèles de l'esprit.

Au niveau populaire, cela a produit une conception erronée des capacités potentielles des machines.

Au niveau logique, cela a bloqué les efforts pour représenter les connaissances ordinaires, en présentant cette image inaccessible d'un corpus de "vérités" hors-contexte qui tiennent à peu près par elles-mêmes.

Au niveau intellectuel, cela a empêché de saisir l'idée fondamentale que penser commence d'abord par des plans et des images suggestifs mais défectueux, qui sont petit à petit (quand ils le sont) raffinés et remplacés par de meilleurs."

[Minsky, 1981]

"La confusion entre tâches et activités s'enracine dans l'identification des descriptions avec les concepts, et rend compte de la difficulté de comprendre que les situations sont des constructions conceptuelles, et non des lieux ou des descriptions de problèmes"

[Clancey, 1997a]

Introduction

"Intelligence artificielle : Secteur de l'informatique ayant trait au raisonnement symbolique et à la résolution de problèmes."

[Kodratoff et Barès, 1991, p. 59]

Le travail de thèse que nous allons présenter ici propose un modèle simple de représentation des connaissances opératoires et un outil pour faciliter la correction et l'enrichissement des Bases de Connaissances (BC) bâties à l'aide de ce modèle. Notre modèle de représentation des connaissances pourrait tout aussi bien être classé dans la catégorie des représentations procédurales que dans celle des représentations déclaratives.

Or il est souvent fait abusivement l'équation suivante : intelligence artificielle (IA) = connaissances déclaratives. Nous allons débuter ce mémoire en défendant que notre approche, bien que ne recherchant pas la déclarativité, relève bien de l'IA. D'abord parce que notre approche se situe dans la continuité directe d'un certain nombre de travaux en IA, et surtout parce que nous nous retrouvons parfaitement dans l'objectif général de l'IA : la conception d'outils intelligents.

Les chercheurs en IA ont toujours cherché à définir leur discipline, soit en explicitant des objectifs généraux fixés à plus ou moins long terme, soit en cherchant simplement à distinguer l'IA des autres disciplines de l'informatique.

L'approche qui consiste à définir l'IA autour de ses objectifs amène bien vite sur le terrain philosophique. Des questions fondamentales, auxquelles il n'est jamais

inutile de réfléchir, se posent, comme “qu'est-ce que l'intelligence ?”, “A quoi peut aboutir à long terme l'IA ?”. Les définitions retenues pour qualifier l'intelligence artificielle sont souvent bâties à partir des réponses données à ces questions et elles donnent souvent lieu à des controverses.

La notion de *rationalité*, telle qu'elle est par exemple formulée par [Newell, 1982], est une notion assez bien acceptée par les chercheurs en IA pour caractériser les comportements intelligents. C'est là un premier type de définition :

“... nous adoptons le point de vue que l'intelligence touche à l'action rationnelle. Idéalement, un agent intelligent effectue la meilleure action possible dans une situation”

[Russell et Norvig, 1995, p. 27]

Des approches plus pragmatiques, comme celle d'Alan Türing proposant un test pour détecter objectivement si une machine est intelligente, permettent de définir l'IA sans définir l'intelligence : “L'IA est la technique qui permet de construire des systèmes effectuant des tâches qu'on qualifierait d'intelligentes si elles étaient effectuées par des personnes”. Ce second type de définition a le mérite d'être assez consensuel. Mais il ne donne pas de frontière bien nette entre informatique “classique” et intelligence artificielle.

“En quoi l'intelligence artificielle est-elle différente des techniques traditionnelles de l'informatique ? L'informatique s'est intéressée à des méthodes algorithmiques dont l'élaboration est guidée par des critères d'efficacité. L'IA s'intéresse à des méthodes de représentation de connaissance et de raisonnement dont l'élaboration est basée sur des critères d'explication.”

[Boy, 1988, p. 2]

On arrive ainsi à un troisième type de définition, dans lequel IA et informatique “algorithmique” sont opposés:

“Tout problème pour lequel aucune solution algorithmique n'est connue, relève a priori de l'IA”

[Laurière, 1986, p. 2]

L'argument de Jean-Louis Laurière est que l'IA prend la relève lorsque les techniques algorithmiques classiques ne donnent pas de résultat. Dans le prolongement de cette idée, il a d'ailleurs été proposé d'abandonner l'intitulé “Intelligence Artificielle” et de redéfinir le sigle IA par “Informatique Avancée”, ce qui aurait pour effet d'éviter les malentendus liés au terme “intelligence”.

Un quatrième type de définition bâtit l'opposition entre informatique

algorithmique et intelligence artificielle sur l'opposition entre le “procédural” et le “déclaratif” :

“Nous dirons qu'une représentation est procédurale lorsque la connaissance que l'on veut lui faire exprimer est représentable algorithmiquement. (.../...)

Par contre, si la connaissance du domaine étudié n'est exprimable que sous forme de déclarations, la représentation correspondante sera appelée représentation déclarative. Ces déclarations sont stockées sous la forme de structures symboliques accessibles par des procédures générales capables de traiter la connaissance ainsi exprimée.

Cette différenciation des représentations de la connaissance permet de distinguer l'informatique ‘classique’ de l'IA.”

[Boy, 1988, p. 6]

L'écueil dans cette démarche qui consiste à chercher à délimiter l'IA comparativement aux autres champs de l'informatique, est de finir par restreindre l'IA à une technique précise, un formalisme ou un paradigme dominant à un moment donné. En l'occurrence ici, c'est le paradigme déclaratif qui est identifié à l'IA. La règle d'inférence ou règle de production est l'objet par excellence du paradigme déclaratif. Celui-ci est fondé sur une hypothèse de complète indépendance entre les éléments de connaissances “déclarés” : les règles. Or dans la pratique, cette hypothèse est très rarement vérifiée. Et si l'on admet que le fondement de l'approche basée sur les règles est fissuré, rien n'oblige à rester accroché à ce formalisme...

Et comme le rappelle Jean-Gabriel Ganascia, l'IA doit procéder d'un “pragmatisme cognitif”...

“... en ce sens qu'aucune connaissance d'aucun ordre ni aucune procédure d'obtention n'est privilégiée. Seule importe l'évidence immédiate qui découle des simulations effectuées. Autrement dit, l'origine pratique des représentations et des connaissances ressortit à un libre choix. Cependant, pour être libre, ce choix n'en est pas moins arbitraire, il dérive des questions que l'on désire traiter et des sources d'inspiration que l'on s'accorde”

[Ganascia, 1990, p. 243]

Pour notre part, nous en resterons donc à une définition assez générale de l'IA (du premier type, c'est-à-dire en s'appuyant sur la notion de rationalité, ou, de manière beaucoup plus floue, selon une définition du second type), et nous ne chercherons pas à délimiter une frontière plus stricte entre IA et informatique dite classique.

Ces principes étant maintenant précisés, nous pouvons formuler l'objectif général de notre travail, qui consiste à **construire des Bases de Connaissances (BC) opératoires évolutives** :

- *opératoires* au sens où l'expertise représentée dans la BC est un *savoir-faire* concret, par opposition à des connaissances *substantives*, c'est-à-dire qui portent uniquement sur l'état des choses. Nous préférons utiliser les termes "connaissances opératoires / connaissances substantives" plutôt que "connaissances procédurales / connaissances déclaratives", lorsque nous parlons de la *nature* des connaissances, sans faire aucune hypothèse sur le *formalisme* utilisé pour représenter ces connaissances. En effet, "procédural" renvoie au vocabulaire algorithmique et "déclaratif" au paradigme du même nom.

- *évolutives* au sens où l'expertise représentée dans la BC peut toujours être corrigée ou enrichie. Notre objectif est que la BC construite puisse toujours être facilement révisée.

Nous voudrions insister sur le statut fondamental des connaissances opératoires dans l'activité cognitive. Toute connaissance a toujours au départ un statut opératoire, puisque c'est l'activité qui fonde la cognition. "Pour aller à Romans, sortir de l'autoroute à Tournon" est un exemple de connaissance opératoire. On peut considérer que cette connaissance "initiale" peut ensuite être abstraite et stockée différemment, sous une forme substantive ("La sortie 'Tournon' est la sortie de l'A7 la plus proche de Romans"). Mais cette connaissance substantive est toujours destinée en fin de compte à être "utilisée" ou "appliquée" : elle redevient connaissance opératoire de manière fugace, à l'instant même de cette "application". Autrement dit, le problème de l'acquisition, de la représentation et de la maintenance des **connaissances opératoires** est à notre avis un sujet central pour l'IA.

Faciliter la révision de BCs évolutives est un aspect primordial de notre objectif. Sans refuser en bloc le concept de déclarativité, nous pensons que les objets d'une Base de Connaissances ne sont jamais indépendants et forment un réseau dans lequel tous les liens entre objets doivent être explicités, de manière à rendre d'une part la BC compréhensible, et pour d'autre part en faciliter la maintenance.

Ce travail propose donc une solution d'architecture et de représentation des connaissances, qui cherche à répondre à ces objectifs et ces principes. L'idée centrale est de construire les BCs opératoires autour de la notion de **situation**. Chaque situation est représentée explicitement dans la BC par un objet appelé "nodule de situation", qui prend place au sein d'un réseau de nodules de situations.

Le concepteur de la BC associe à chaque nodule une **action** nécessairement exécutée dans cette situation. L'exécution de l'action renvoie un ou plusieurs faits, qui, stockés dans une Mémoire de Travail, servent ensuite à choisir un nouveau nodule de situation. Les connaissances nécessaires au choix du nouveau nodule sont stockées localement, sous forme de règles associées au nodule. Le moteur qui exploite la BC active donc un nodule initial, qui lui-même en active un autre, etc. Le processus se termine lorsque le nodule activé ne peut choisir de nodule successeur. Le dernier nodule activé représente l'interprétation finale de la situation et contient l'action adéquate qu'il convient d'exécuter pour la tâche et l'environnement actuel. A moins que la BC ne nécessite une révision...

Ainsi, le modèle simple que nous proposons, fondé sur les notions de situation et d'action, permet à l'utilisateur d'avoir une vision claire et explicite du système en fonctionnement. Cette visibilité et la compréhensibilité de la BC qu'elle permet sont des atouts pour la révision de la BC.

Notre approche prend donc délibérément le parti-pris de **l'explicite** : les savoir-faire sont représentés de manière synthétique et complète, c'est-à-dire exhaustivement (tous les cas de figure qui peuvent se présenter sont pris en compte) et précisément (la “marche à suivre” pour traiter ces cas de figure est décrite exactement par le concepteur).

Or, de même que l'on conteste parfois l'appartenance des connaissances “procédurales” au domaine de l'IA, l'expertise complètement explicitée ne recèle plus aucun mystère et peut alors paraître trop simple pour toujours relever de l'IA...

C'est là une sorte d'idée reçue qui associerait l'intelligence à quelque chose de magique, du moins à quelque chose de complexe. Peut-être est-ce là la vertu des systèmes experts possédant des bases de règles de grande taille, que d'être capables de sélectionner la bonne connaissance au bon moment “comme par enchantement” et de perpétuer ainsi une certaine illusion...

A l'opposé, lorsque l'on est capable de suivre à la trace, ou mieux, d'anticiper, le raisonnement du système, on aura des difficultés à qualifier le comportement du système d'“intelligent” et on sera même enclin à trouver que l'emploi du terme “raisonnement” est exagéré. C'est sans doute la croyance plus ou moins explicitée que l'intelligence est le propre de l'homme qui pousse chacun à considérer qu'une tâche, dès lors qu'elle est automatisée ou automatisable, ne peut alors plus être intelligente...

Nous voudrions donc nous inscrire en faux d'abord contre l'idée qu'intelligence est synonyme de complexité. Dire de quelqu'un qu'il est intelligent ne signifie heureusement pas toujours : “Je n'ai rien compris de ce qu'il a dit, mais il a

l'air de bien s'y connaître. Je lui fais confiance". La **compréhensibilité** est une propriété à notre avis essentielle de l'intelligence[1].

Pour schématiser notre point de vue, nous préférons faire exploser la grosse boîte noire, qui "impressionne", complexe parce qu'unique, en une multitude de petites boîtes transparentes, chacune simple, triviale, spécialisée, mais pas moins "intelligente". Alors que la boîte noire doit *expliquer* son comportement, la petite taille des boîtes transparentes fait que celles-ci sont parfaitement *lisibles* : il suffit d'examiner directement leur contenu pour comprendre le comportement du système. Cela n'est pas un plaidoyer en faveur du paradigme multi-agents : la compréhensibilité de la petite boîte signifie compréhensibilité du système uniquement parce que le modèle qui définit l'ordonnancement général et le fonctionnement des boîtes, autrement dit le *contrôle*, est simple, uniforme, séquentiel et prévisible.

Un des thèmes centraux de cette thèse est que **révisabilité**[2] et compréhensibilité sont intrinsèquement liées. Également centrale est l'idée que les objets de la BC ne peuvent être compréhensibles que si le concepteur de la BC participe activement à leur création et à leur maintenance. Il est alors logique dans ces conditions que la révision se déroule de manière **coopérative** entre le système (capable d'effectuer des opérations de révision complexes...) et le concepteur, qui doit, selon nous, garder une certaine mainmise sur les objets de la BC.

---

[1] La "lisibilité" du comportement d'un agent fait partie du concept d'agent *rationnel*, souvent synonyme d'agent intelligent.

[2] Nous définissons la *révisabilité* comme la propension d'une représentation (au sens "grain" de connaissance) à être facilement révisée. *Facilement* signifie que la révision de cette représentation est une tâche peu coûteuse (en effort et en temps) pour le concepteur.

Guide de lecture

Le chapitre 1, “Une démarche qui aboutit à un modèle”, décrit le cheminement qui nous a mené, partant d'un objectif général (faciliter la construction incrémentale d'une BC opératoire) à construire un modèle centré sur la notion de situation. Dans un premier temps nous expliciterons les objectifs recherchés autour des notions d'incrémentalité (apprendre pas à pas), de compréhensibilité et de “révisabilité”. Dans un second temps nous examinerons les différentes “réponses” qui s'offrent à notre projet : nous ferons ainsi un tour d'horizon des différentes techniques utilisées et paradigmes fondateurs (de la règle d'inférence à la cognition située), qui sont autant d'influences pour affiner notre projet — auquel nous donnerons finalement le nom de “révision située” — et pour inspirer la construction de notre propre réponse. Dans un troisième temps nous décrirons le raisonnement qui nous a mené à l'élaboration d'un modèle original de représentation des connaissances. Nous montrerons comment le modèle Situation-Action, extension du modèle Stimulus-Réponse, nous amène, pour représenter les savoir-faire au sein d'une BC, à proposer un objet nommé “nodule de situation”.

Le chapitre 2, “Un outil et une méthodologie d'acquisition des connaissances” présente l'outil d'acquisition des connaissances bâti à partir du modèle défini au chapitre 1. Cet outil a été élaboré pour une application industrielle : la conception de dialogues personnalisés pour le serveur télématique des Caisses de Prévoyance et de Retraite de la S.N.C.F.. Le modèle théorique du chapitre 1 a été adapté et enrichi à la lumière de contraintes réelles et d'expérimentations concrètes. Cet aller-retour entre une théorie à prétention générale et une application concrète a été une réalité tout au long de ce travail : il y a bien eu à la fois des efforts pour chercher à généraliser au mieux l'expérience pratique et inversement pour chercher à valider et à affiner expérimentalement le modèle sous-jacent. Nous avons choisi de présenter le modèle général avant l'application pratique, non seulement pour la clarté de la présentation, mais aussi parce que cela correspond vraiment à l'histoire de ce travail : d'une intuition vers une application.

L'expérience de construction de Bases de Connaissances à l'aide de notre modèle nous a conduit à proposer une méthodologie empirique d'acquisition des connaissances autour de la notion de situation. Cette méthodologie est décrite dans ce chapitre 2, ainsi que des premières évaluations de l'outil développé. Des perspectives de développement d'EDINOS sont également dégagées, d'une part autour des opérateurs de restructuration de la BC, et d'autre part, pour faire face au

problème de la réutilisation des composants de la BC. Une discussion sur l'intérêt de l'approche incrémentale d'acquisition des connaissances clôt ce chapitre.

Le chapitre 3, “REVINOS : un outil de révision coopérative autour de la représentation par nodules de situation” est important puisqu'il s'attaque à l'objectif principal de notre travail : donner au concepteur de la BC les moyens (techniques et “conceptuels”) de facilement réviser la BC. L'outil REVINOS complète la méthodologie du chapitre 2 en guidant le concepteur tout au long de la session de révision. Le dialogue ci-dessous résume, de manière schématique, un exemple de session de révision coopérative ayant lieu entre l'utilisateur (concepteur de la BC) et le système :

REVINOS : Dans ce cas précis, j’effectue l’action A.

UTILISATEUR : Non, pour ce cas-là, c’est l’action B qu’il faut effectuer.

REVINOS : D'accord... Dans ces types de cas similaires [Description des types de cas], faut-il effectuer également l’action B ?

UTILISATEUR : Oui.

REVINOS : Très bien, la révision est terminée. J'ai corrigé la BC conformément aux indications données.

Le concepteur n'a ainsi pas besoin de corriger les règles de choix des nodules de situation : l'outil de révision s'en charge lui-même, laissant le concepteur se concentrer sur les choix “conceptuels” qui sont beaucoup plus importants. Les corrections qu'effectuent le système automatiquement sont validées au moyen d'exemples générés par le système, acceptés ou rejetés par le concepteur.

Le chapitre 4, “Quelques représentations similaires aux nodules de situation” revient sur le modèle de représentation des connaissances par nodules de situation. Une sorte de validation “théorique” de l'intérêt du modèle consiste à montrer que ce modèle partage beaucoup de points communs avec de nombreux autres formalismes pour représenter des savoir-faire en IA. La détection de similarités (mais aussi de différences notables) avec de multiples formalismes dans des domaines aussi variés que le traitement de la langue naturelle, l'acquisition des connaissances, les travaux sur le contexte, la robotique située, la surveillance de processus, et même la psychologie cognitive, peut être interprétée positivement comme le signe que notre modèle s'appuie sur des intuitions qui ne sont pas isolées, mais partagées avec un certain nombre de travaux récents, qui répondent chacun à des préoccupations et à des besoins réels dans leurs domaines respectifs.

Chapitre 1 :

Une démarche qui aboutit à un modèle

L'évolution technologique peut sembler vertigineuse. Dans le domaine de l'informatique en particulier, on peut parvenir à quantifier cette évolution lorsqu'elle concerne la fréquence d'horloge des microprocesseurs, la taille des composants ou la capacité des mémoires. Gordon Moore, cofondateur de la société Intel fit en 1965 la constatation suivante : la densité en transistors des microprocesseurs double tous les deux ans. Autrement dit, tous les deux ans une nouvelle puce apparaît, approximativement deux fois plus puissante que la précédente. Plus de trente ans après, cette constatation impressionnante continue à se vérifier. Elle est devenue connue sous le nom de “loi de Moore” et sert à fonder un grand nombre de prévisions en matière de performance d'ordinateurs. L'informatique a énormément profité de ces développements qui ont permis la miniaturisation et la démocratisation des ordinateurs.

Mais parallèlement et conséquemment à cette évolution quantitative, a lieu en informatique une évolution “qualitative” qui, elle, si elle n'est pas mesurable, n'en est pas moins tout aussi importante. Car l'informatique génère régulièrement des concepts neufs qui refondent la manière de travailler aussi bien au niveau individuel (souris, multifenêtrage, hypertexte,...) qu'au niveau collectif (messagerie, collecticiel,...). Ces nouvelles “technologies intellectuelles” ([Lévy, 1990]), ces

nouveaux concepts, voire ces nouveaux "paradigmes" ([Kuhn, 1970]), influent sur les façons de concevoir, de s'organiser, de programmer, et même, peut-on dire, de penser puisqu'ils influencent la construction des représentations mentales.

L'apparition d'un nouveau concept ou d'un nouvel outil technologique a de multiples répercussions et applications qui ne sont pas toujours immédiatement perçues et mises en oeuvre. Dans un contexte d'évolution technologique rapide, il n'est jamais inutile de rechercher de quelles manières les nouveaux concepts et outils, et leurs combinaisons, peuvent permettre d'apporter des réponses nouvelles à des objectifs qui eux aussi ont pu évoluer.

Le propre du travail de recherche c'est de prendre le temps et le recul nécessaire pour élaborer de nouvelles propositions, en s'appuyant sur les concepts et outils utilisés habituellement, et éventuellement en les remettant en question. C'est autant la reconnaissance d'un héritage que sa critique qui permet d'avancer : ni l'un ni l'autre n'est à lui seul suffisant.

Dans le domaine de l'IA en l'occurrence, la logique formelle est le paradigme fondateur[1]. Et l'objet associé "règle d'inférence" a longtemps été et demeure la technologie cognitive dominante pour construire des Systèmes à Bases de Connaissances. En tant que telle, elle est critiquée, parce que pris comme référence, à chaque fois qu'un nouveau concept cherche à creuser son trou. Tel est par exemple le cas lorsque [Minsky, 1981] introduit le concept de "frame" (cadre), ou lorsque [Riesbeck et Schank, 1989] présentent le raisonnement à partir de cas, ou encore lorsque [Smolenski, 1992] défend l'approche connexionniste.

En évitant de tomber dans la remise en question généralisée ("jeter le bébé avec l'eau du bain") ou dans la recherche de nouveauté à tout prix, ce sont des objectifs de recherche généraux qui vont complètement guider notre travail. Nous allons prendre le temps et la liberté de définir ces objectifs propres de recherche. Dans un second temps, nous passerons rapidement en revue le riche héritage qu'offre l'IA pour répondre à ces objectifs. Et notre insatisfaction toute relative nous conduira à risquer la proposition d'un modèle simple et intuitif, combinaison d'idées et de concepts puisés dans "l'héritage".

1.1 Les objectifs de recherche

1.1.1 Prescriptivité

[1] A tel point qu'il est souvent nécessaire de rappeler que l'IA ne se réduit pas à la logique ([Ganascia, 1990]). "La logique est simplement une représentation de la connaissance. Ce n'est pas la connaissance elle-même, mais une structure au niveau symbolique" ([Newell, 1980, p. 110]). A. Newell a trouvé nécessaire de faire ce type de "rappel"...

Le but général de notre travail est de permettre à un utilisateur, que nous appellerons *concepteur*, de **représenter des savoir-faire** dans une Base de Connaissances (BC). Nous emploierons tour à tour les termes *connaissances opératoires* ou *savoir-faire* pour désigner ce type de connaissances qu'on peut définir comme étant des réponses à la question "comment faire ?"(*know-how* en anglais).

L'action de représenter ce type de connaissances dans un système informatique est une *spécification*. Autrement dit, ces connaissances opératoires représentées correspondent à des **prescriptions** données par le concepteur. Celui-ci veut spécifier des comportements du système pour des situations données.

Le psychologue Jean-François Le Ny distingue représentations naturelles (réelles) et représentations rationnelles (prescriptives) ([Le Ny, 1989]). C'est uniquement ce second type de représentations que nous cherchons à transcrire dans un Système à Bases de Connaissances (SBC). Nous ne sommes donc pas intéressé par des modèles psychologiques réalistes, mais par des modèles prescriptifs ou normatifs.

De plus, cet objectif de prescriptivité exclut les modèles non déterministes susceptibles de générer des comportements différents sur des situations identiques, comme par exemple les approches stochastiques ou probabilistes, ou les approches fondées sur la notion "d'émergence".

1.1.2 Incrémentalité

On peut très schématiquement distinguer deux approches dans la construction de Systèmes à Bases de Connaissances (SBC).

1) Une première approche s'attache à construire des systèmes **définitifs**, c'est-à-dire des systèmes dits "experts", définitivement validés. Les connaissances de ces systèmes sont censées être "correctes et complètes". Dans son domaine de compétence, le SBC doit être complètement fiable et efficace. Pour des applications critiques (supervision de réseaux, surveillance de centrale nucléaire, etc.), c'est-à-dire lorsque la moindre erreur dans l'exploitation du système possède un coût important, il est indispensable que le système réponde à ces caractéristiques.

Mais certains domaines se prêtent moins facilement que d'autres à la construction de systèmes définitifs et à une formalisation complète "a priori". Ainsi, s'il est répandu de construire des systèmes définitifs pour des problèmes de recherche dans des espaces d'états (par exemple pour des problèmes "artificiels" comme les problèmes de jeux), d'une manière plus générale les savoir-faire concrets

se laissent difficilement représenter complètement. On peut avancer une première explication à la différence qu'il peut y avoir entre le domaine des jeux et les savoir-faire concrets que possèdent les experts dans leur domaine professionnel. Dans le domaine des jeux, il est possible (et même nécessaire) de donner explicitement et à l'avance, les caractéristiques des états-buts recherchés : par exemple “il y a échec et mat lorsque le Roi ne peut fuir sans se mettre en échec, et si l'échec ne peut être paré d'une quelconque façon”. Il est plus aisé de fabriquer des systèmes définitifs lorsque “ce qu'est un problème résolu” peut être aussi précisément caractérisé : la résolution de problème devient “choix d'un parcours dans un espace d'états”. Or dans de très nombreux domaines concrets, il est très difficile de donner des règles définissant les caractéristiques générales du “problème résolu”. De plus, même lorsqu'il y a une manière objective de savoir si le problème est résolu, la manière pour y arriver est souvent déterminante pour valider la solution proposée. En résumé, on peut construire assez facilement des systèmes définitifs pour une classe assez restreinte de domaines, dans lesquels les caractéristiques des états solutions sont connues au départ, et toutes les contraintes sur la manière de parvenir à la solution ont été explicitées. Il est beaucoup plus difficile de fabriquer des systèmes définitifs dans les autres domaines.

2) La seconde approche consiste à admettre que tout Système à Bases de Connaissances comprend des savoir-faire limités et potentiellement erronés. Selon cette approche, un SBC doit être considéré comme un système **évolutif** (ou provisoire). La première caractéristique d'un système évolutif, est d'être **opérationnel**, c'est-à-dire d'être capable de d'ores et déjà donner des solutions, sans attendre une étape future de validation.

Nous nous situons dans un cadre où le système interagit avec un utilisateur, soit pour lui proposer des solutions, soit pour intégrer de nouvelles solutions (l'utilisateur joue le rôle de tuteur). Dans ce contexte, commun aux travaux en acquisition des connaissances, mais différent par exemple des travaux en robotique située, un système évolutif doit logiquement être pourvu :

- d'un module d'explication de manière à communiquer à ses utilisateurs le détail de ses raisonnements : si l'on ne fait plus confiance aveugle au système parce que celui-ci n'est plus définitif, le système doit être capable de “s'expliquer”.

- d'un module de révision coopérative, de manière à faciliter la correction et l'enrichissement de la BC au fur et à mesure de la découverte de cas de figure nouveaux ou pour lesquels le système réagit de manière erronée.

La construction de ces systèmes du second type ne peut donc se faire de manière incrémentale que si ces derniers possèdent ces modules d'explication et de révision.
Le cadre de travail dans lequel nous nous situons est le suivant. A un instant *t*, le système opérationnel est confronté à un cas de figure qu'il sait résoudre ou non. Nous faisons l'hypothèse de la présence d'un utilisateur-expert, compétent dans le domaine, disponible pour :

1) détecter les erreurs ou blocages de la BC à partir de l'examen du comportement du SBC “en exploitation” ou d'un module de simulation.

2) collaborer ensuite à la correction de la BC.

Le but est évidemment au fur et à mesure des enrichissements apportés (de l'expérience accumulée) d'augmenter la fiabilité du système et d'étendre sa compétence.

Bien entendu les deux approches ne sont pas antinomiques. Il est évidemment possible de construire incrémentalement des systèmes évolutifs, avant de les valider définitivement. Mais dans l'histoire de l'IA, il semble que les SBC aient d'abord été vus comme des systèmes “définitifs” : lorsque l'on pense pouvoir représenter exhaustivement la connaissance d'un expert dans un ordinateur, les problèmes vont être posés en termes de transfert de connaissances et de validation. Historiquement, la tâche de transfert des connaissances a changé de nom et est devenue véritable tâche de modélisation, donc de conception et le problème de la maintenance des BCs est devenu une préoccupation majeure. Mais la vision de l'acquisition des connaissances comme un "goulot d'étranglement", procède de cette vue selon laquelle les connaissances peuvent être acquises complètement et qu'alors il serait facile d'obtenir de manière définitive, par un processus monotone, un système “correct et complet”.

L'objectif général de notre démarche est donc de chercher à faciliter **la construction incrémentale** d'un SBC évolutif. Etre capable de construire une BC incrémentalement, donc de réviser à tout moment cette BC, est un problème de recherche important, voire central si l'on adhère à la maxime suivante de Karl Popper :

“Toute connaissance acquise, tout apprentissage, consiste en la modification (éventuellement le rejet) d'une forme de connaissance, ou disposition, qui était là précédemment”

[Popper, 1979, p. 71]

Une première conséquence de cette recherche d'incrémentalité de la construction de la BC est la nécessité d'une certaine "**révisabilité**" de cette BC, c'est-à-dire une facilité à être corrigée et enrichie. Comme nous allons l'expliciter dans la section suivante, un troisième facteur important à considérer est la **compréhensibilité** de la BC. Incrémentalité, révisabilité et compréhensibilité sont à notre avis étroitement liés.

1.1.3 Compréhensibilité et révisabilité

La construction d'une BC peut être faite automatiquement à partir d'une somme de cas résolus (ensemble d'exemples) connue au départ : c'est ce que permettent les systèmes d'apprentissage automatique comme INDUCE ([Michalski, 1983]), ID3 ([Quinlan, 1986]), FOIL ([Quinlan, 1990]), OGUST ([Vrain, 1990]), KBG ([Bisson, 1993]), ...

Mais se posent alors des problèmes de compréhensibilité :

"Il y a une seule chose commune à toutes les applications industrielles de l'apprentissage automatique : les utilisateurs exigent la compréhensibilité des résultats de l'algorithme d'apprentissage, et plus la compréhensibilité est grande, plus ils sont enthousiastes"

[Kodratoff, 1994, p. 83]

La compréhensibilité des connaissances apprises a longtemps peu intéressé les chercheurs en apprentissage, ceux-ci s'intéressant à améliorer l'efficacité des algorithmes d'apprentissage. Une des raisons avancée par [Kodratoff, 1994] est que la compréhensibilité n'étant pas une entité mesurable, elle échappe à une approche purement scientifique, qui a besoin de critères objectifs d'évaluation et de comparaison.

Si le problème de la compréhensibilité de la BC se pose pour l'apprentissage automatique, il se pose dans les mêmes termes pour la révision. Car de la même façon qu'en apprentissage, on peut proposer qu'une BC "évolutive" soit révisée uniquement à partir de la description d'un ensemble d'exemples, sans l'intervention d'un utilisateur (par exemple, SEEK2 [Ginsberg, 1988a]). Contrairement aux algorithmes d'apprentissage automatique, les algorithmes de révision s'appuient sur une BC déjà existante, qui éventuellement possède déjà des propriétés de compréhensibilité. Mais même dans ce cas, où la BC est compréhensible avant la révision, le problème de la compréhensibilité se posera toujours pour les connaissances nouvellement apprises...

Notre point de vue est que nous ne pourrons résoudre ces problèmes de compréhensibilité qu'en plaçant l'utilisateur-concepteur au cœur du processus d'apprentissage ou de révision.

La démarche de construction incrémentale doit faciliter la compréhensibilité puisqu'on peut alors permettre à l'utilisateur à chaque étape de s'approprier les objets de la BC. On ne peut obtenir à notre avis une BC compréhensible que si l'utilisateur maîtrise complètement la création et la modification des objets de la BC, s'il peut commenter ces objets au fur et à mesure de la construction de la BC. La révision doit être le fruit d'une **coopération** entre la machine et l'utilisateur. Celui-ci doit garder un rôle central dans un processus, la construction incrémentale de la BC, qui met aussi bien en jeu des activités de modélisation, que des activités de révision.

Si la compréhensibilité est facilitée par l'incrémentalité, l'inverse est également vrai[2]. En effet la révisabilité dépend de la compréhensibilité de la BC... Nous verrons tout au long de ce mémoire que ces deux caractéristiques sont étroitement liées l'une à l'autre. La compréhension est indispensable à la révision pour permettre à l'utilisateur de spécifier des opérations précises sur les objets de la BC, au cours de la correction.

En conclusion, si l'on veut faciliter la construction incrémentale de SBC, il faut chercher à faciliter au mieux :

- la **compréhensibilité** de la BC par l'utilisateur,
- et la **révision coopérative** de la BC.

1.1.4 Interruptibilité

Une troisième caractéristique est nécessaire à une construction incrémentale, de la BC. Nous l'appellerons “contrainte d'interruptibilité”. Elle consiste à exiger que l'utilisateur puisse **à tout moment interrompre** l'exécution du système, pour :

- dans un premier temps comprendre exactement “ce qui se passe”, c'est-à-dire comprendre l'étape du raisonnement en cours. La propriété de compréhensibilité couplée à cette contrainte d'interruptibilité entraîne une **exigence de transparence,** qui doit se traduire par une représentation **explicite** des connaissances. De plus, la **séquentialité** du raisonnement est nécessaire pour permettre la lisibilité des états du système lors d'interruptions inopinées du système en fonctionnement.

---

[2] Nous avons pris l'incrémentalité comme objectif principal, mais en partant avec la compréhensibilité pour objectif initial, nous aurions fait le même chemin...

- éventuellement dans un second temps, pouvoir spécifier précisément ce qu'il convient d'effectuer si le comportement du système ne donne pas satisfaction.

La propriété de révisabilité de la BC entraîne une **exigence de précision** : l'utilisateur doit pouvoir spécifier l'exécution d'une action précise dans une situation précise. Cette situation précise peut être détectée par l'utilisateur à n'importe quel moment au cours d'une exécution ou d'une simulation du système.

Cette contrainte d'interruptibilité exclut les systèmes type “boîte noire”, où le processus qui transforme des entrées en sorties est un processus opaque, qui n'est pas prévu pour être présenté et détaillé à l'utilisateur. L'interruption d'une boîte noire en fonctionnement ne peut fournir par définition aucune information *explicite*. D'autre part l'utilisateur n'a aucun contrôle sur le cheminement effectué à l'intérieur de la boîte noire : on ne lui permet pas de *préciser* des contraintes sur la manière de résoudre le problème. La seule exigence que peut formuler l'utilisateur à une boîte noire est une exigence sur les résultats (donner les bonnes sorties en fonction des entrées).

Nous retiendrons donc que notre modèle, pour répondre à la contrainte d'interruptibilité, doit intégrer une **représentation explicite, séquentielle et précise du raisonnement**.

1.1.5 Le modèle de représentation des connaissances

Nous avons spécifié dans la section 1.1.2 que la BC construite pour notre système évolutif devait être constamment **opérationnelle**. Le savoir-faire du domaine qui nous intéresse doit pouvoir être exprimé dans le formalisme d'un modèle de représentation des connaissances.

Ce modèle de représentation des connaissances qui pourrait permettre de répondre à nos objectifs de prescriptivité, d'incrémentalité, de compréhensibilité, de révisabilité et d'interruptibilité doit posséder selon nous un certain nombre de caractéristiques que nous allons détailler.

En premier lieu, il apparaît que la **simplicité** et **l'uniformité** du modèle sont des critères importants de compréhensibilité. Il est plus difficile à un utilisateur de s'y retrouver lorsque chaque type de connaissances est exprimé par un formalisme différent. D'autre part, le recours à une structure de contrôle unique, comme par exemple l'envoi de messages dans les langages à objets, permet d'avoir une représentation des connaissances homogène et compréhensible.

D'autre part, une structure complexe conçue pour être appliquée à un domaine particulier d'expertise perd de son intérêt si elle n'est pas réutilisable.

L'investissement en terme d'effort de compréhension de l'utilisateur est d'autant moins rentable si le modèle est complexe et s'il ne possède guère de **généricité**.

Ce nécessaire compromis entre simplicité et universalité est proche d'un deuxième antagonisme : simplicité contre expressivité ou précision. En effet, le modèle doit conserver parallèlement l'**expressivité** nécessaire à l'expression de connaissances qui sont dans la tête du concepteur bien déterminées et non ambiguës (exigences de précision et de prescriptivité). Comme le souligne Tom Gruber qui étudie l'acquisition des connaissances qu'il appelle *stratégiques* :

> *"Il y a une tension entre l'exigence de fournir à l'utilisateur un langage compréhensible et cependant suffisamment puissant pour mettre en oeuvre la stratégie"*
>
> [Gruber, 1989a, p. 301]

La simplicité du modèle ne doit pas se payer en perte de pouvoir d'expression. Or nous partons du principe que les connaissances sont toujours localisées, au sens où leur domaine de validité est limité. La reconnaissance de cette **localité** des connaissances va de pair avec notre démarche d'acquisition incrémentale, où l'on procède cas par cas et où l'on cherche à délimiter avec précision l'étendue d'application des connaissances nouvellement acquises. On peut qualifier de **prudente** cette approche, dans la mesure où l'on préfère sous-généraliser les connaissances plutôt que de faire des pas inductifs non validés. Le choix du modèle de représentation des connaissances est un engagement relativement à ce problème. Si le concepteur utilise par exemple pour rechercher la concision, une représentation conçue pour exprimer des règles générales, il aura naturellement tendance à définir des principes généraux, donc à risquer la surgénéralisation.

Notre démarche a donc été esquissée : pour faciliter la construction incrémentale de la BC, nous recherchons un modèle prescriptif de représentation de savoir-faire qui soit compréhensible et révisable à tout moment, c'est-à-dire qui soit notamment simple, uniforme, explicite, séquentiel, précis plutôt que concis. Le schéma de la figure 1.1 résume ces objectifs.

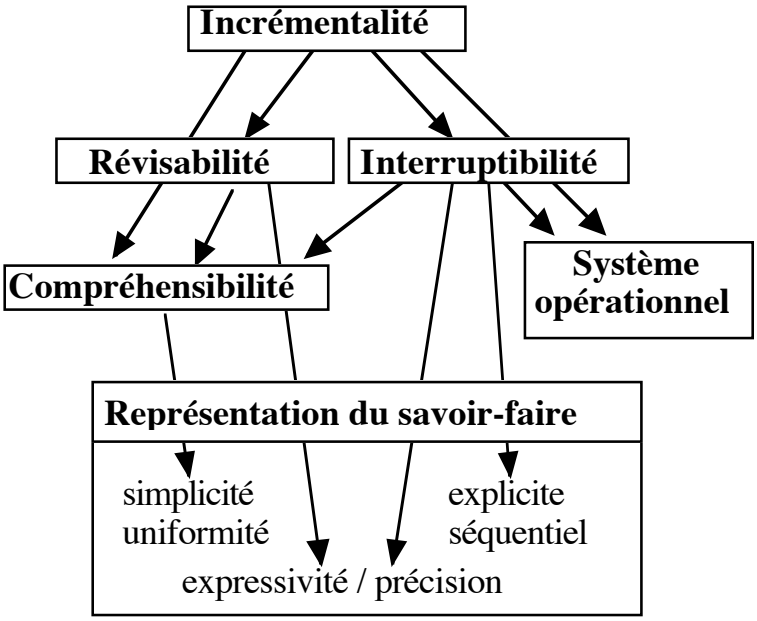


Figure 1.1 : Nos objectifs

1.2 L'héritage (un point de vue)

Dans cette section, nous passons en revue les différents concepts et outils qui s'offrent à nous pour répondre à nos objectifs. Nous allons voir qu'aucun de ces concepts et travaux ne suffit à lui seul pour satisfaire les contraintes que nous nous sommes fixées. Mais nous trouverons dans cet héritage un foisonnement de concepts et d'idées qui détermineront la construction de notre modèle. Cet “état des lieux” est décomposé en trois parties. D'abord nous retiendrons un certain nombre de concepts qui nous semblent intéressants, voir incontournables, pour la tâche que nous nous sommes assignée. Il s'agit des notions de règle, de tâche et de rôle, de cas et d'activation. Ensuite nous présenterons des idées et des outils qu'on peut difficilement résumer à un concept unique, et qui correspondent plutôt à un ensemble de pratiques et de travaux de recherche. Dans cette catégorie, nous présenterons les environnements de programmation orientée objet, et les systèmes apprentis. Enfin, cette section est close par la présentation du courant d'idée, dit “cognition située”, qui, particulièrement sous l'impulsion de William Clancey, apporte à l'IA un questionnement critique sur ses fondements, ses pratiques et ses objectifs. Un certain nombre de ces critiques stimulent et renforcent la démarche que nous nous sommes proposée. Dans la discussion, nous serons amené à resituer notre démarche par rapport aux autres démarches possibles en IA. Nous

proposerons le terme “révision située” pour qualifier le cadre de recherche dans lequel s’inscrit notre démarche.

1.2.1 Des concepts

1.2.1.1 La notion de règle

“Par essence, l'approche basée sur les règles rend implicite ce qui devrait être explicite (c'est-à-dire le flux du contrôle), et rend explicite ce qui devrait être implicite (c'est-à-dire le contexte)”

[Georgeff, 1985]

L'IA s'enracine dans la logique formelle ([Hofstadter, 1988]). En conséquence, la règle d'inférence a été depuis le départ la brique de base pour la construction de Systèmes à Base de Connaissances (SBC). Une règle d'inférence est un objet qui associe des prémisses d'une part, et des actions ou conclusions d'autre part. Elle fournit un formalisme de représentation des connaissances simple, uniforme et explicite, qui semble répondre naturellement aux objectifs de déclarativité que résume Jean-Gabriel Ganascia ainsi :

“(...) dans le paradigme déclaratif, la programmation se résume à une description formelle de la connaissance. Dans la mesure où ces connaissances seront intelligibles, les programmes résultant seront transparents; dans la mesure où elles seront granulaires, les programmes seront facilement modifiables”

[Ganascia, 1990, p. 48]

La déclarativité, ainsi définie, paraît répondre aux objectifs de compréhensibilité et de révisabilité que nous nous sommes fixés. La connaissance déclarative est exploitée par un moteur d'inférence, dont le comportement doit être suffisamment simple pour être parfaitement connu et maîtrisé par le concepteur. Dans ces seules conditions, le concepteur est en mesure de *comprendre* les connaissances déclarées, parce qu’il sait *comment* celles-ci seront utilisées. Cette architecture a donné lieu à la construction de nombreux systèmes experts, dits de première génération, à la suite des systèmes DENDRAL ([Feigenbaum, 1977]) et MYCIN [Buchanan et Shortliffe, 1984]).

Historiquement, c’est autour du projet MYCIN qu’on retrouve le plus clairement explicité l’objectif de déclarativité. L'idée est que la compréhensibilité des règles vient de leur simplicité et de leur modularité :

"Le but est que chaque règle contienne un morceau de connaissance simple et modulaire, et que soit déclaré explicitement dans la prémisse tout le contexte nécessaire. Puisque la règle utilise un vocabulaire de concepts commun à tout le domaine, elle forme, par elle-même, une déclaration compréhensible d'une parcelle de la connaissance du domaine."

[Davis, Buchanan et Shortliffe, 1977, p. 18]

Une certaine "exigence de précision" (tout le contexte d'application des règles est exprimé dans les prémisses), qui fait partie également de nos objectifs, est ainsi considérée comme une alliée à la compréhensibilité. Au souci d'avoir des "morceaux de connaissances" faciles à examiner, se joint le souci de pouvoir facilement ajouter des règles. Les auteurs insistent sur le fait qu'il y a peu d'interactions entre les règles et que cette indépendance entre les règles facilite la maintenance du SBC.

Or si les systèmes experts dits de première génération ont permis de démontrer qu'il était possible de simuler un savoir-faire dans des domaines circonscrits, ils ont également montré les limites de leur architecture pour la réalisation de l'objectif de déclarativité...

Dans le paradigme déclaratif, les règles sont générales, indépendantes entre elles et elles sont exprimées indépendamment de leurs "utilisations" possibles durant l'accomplissement des tâches. Trois critiques peuvent être faites à ce sujet. La première critique oppose cette indépendance entre les règles et leurs utilisations, à l'objectif de compréhensibilité. La seconde critique porte sur l'indépendance entre les règles. La troisième critique remet en question l'ambition d'avoir des règles générales universellement valables.

Commençons par exposer la première critique. Elle part du présupposé suivant : nous considérons qu'une règle est comprise par un utilisateur dès lors que celui-ci est capable d'en inférer les différentes utilisations schématiques possibles. Pour illustrer cette idée, on peut se rappeler qu'une règle ne peut être validée que si l'on est certain qu'aucune conséquence de l'exploitation de cette règle ne sera mauvaise. La compréhension d'un objet n'est pas une simple contemplation de l'objet, elle nécessite d'envisager un ensemble de conséquences (prendre l'objet comme faisant partie d'un tout[3]).

Comment réconcilier cette conception de la compréhension avec l'objectif de déclarativité ? Grâce au moteur d'inférence : c'est lui qui par définition fait le lien entre la connaissance déclarée et l'exploitation de cette connaissance. Comme nous l'avons déjà souligné, un utilisateur ne pourra par conséquent comprendre une règle que s'il connaît et maîtrise parfaitement le fonctionnement du moteur d'inférence.

[3] Comprendre veut dire étymologiquement "prendre avec".

Prenons un exemple. Comment faut-il comprendre la règle simple R suivante : "si l'on observe des éternuements répétés alors faire l'hypothèse d'un rhume" ? Cette règle est-elle correcte ? Sûrement pas, mais ce que nous voulons montrer, c'est que la manière dont il faut interpréter R va dépendre du moteur d'inférence. Si le moteur fonctionne uniquement en chaînage avant, cette règle permettra d'inférer un rhume à partir du symptôme "éternuements répétés" : il faudra dans ces conditions se soucier par exemple du cas d'une allergie au pollen, qui peut avoir ce même symptôme, pour statuer sur la pertinence de la règle R. Par contre, si le moteur fonctionne en chaînage arrière, la règle peut signifier qu'un éternuement est nécessaire pour diagnostiquer un rhume... Mais pour vérifier cela, il faut alors se pencher sur les autres règles qui possèdent "rhume" en conclusion, pour véritablement comprendre ce que signifie la règle R.

Nous avons développé cet exemple pour illustrer l'idée que la déclarativité n'était a priori possible que si le moteur d'inférence avait un fonctionnement bien connu de celui qui voulait comprendre la règle. Le logicien peut contester notre argumentation en soulignant qu'il n'est peut-être pas possible de statuer sur la pertinence d'une règle prise isolément, mais qu'il faut plutôt demander si la BC prise dans son ensemble est correcte et complète. Si tel est le cas, il devient inutile de se demander si le moteur fonctionne en chaînage avant, arrière, mixte,... L'interprétation de la règle ne dépend plus du moteur d'inférence. Le paradigme logique vient ainsi au secours du paradigme déclaratif... Mais il amène avec lui de grosses contraintes que sont les exigences de cohérence et de complétude...

"... je ne pense pas que la cohérence soit nécessaire ou même désirable dans le développement d'un système intelligent. Personne n'est jamais complètement cohérent. Ce qui est important, c'est comment on traite les paradoxes ou les conflits, comment on apprend à partir des erreurs, comment on contourne les incohérences suspectées"

[Minsky, 1981, p. 126]

Sans développer davantage, la seule contrainte de complétude s'oppose à notre objectif d'incrémentalité : un système évolutif n'est par essence pas complet... Pour reprendre la terminologie de la section 1.1, le cadre de la logique classique se prête à la construction de systèmes *définitifs*, mal à la construction de systèmes *évolutifs*.

"Parce que les logiciens ne sont pas concernés par des systèmes qui seront plus tard élargis, ils peuvent concevoir des axiomes qui donnent uniquement les conclusions voulues"

[Minsky, 1981, p. 125]

"... du point de vue de l'IA, l'inconvénient du "rêve booléen" est que les règles symboliques et la logique qui permet de les manipuler tendent à produire des systèmes rigides et fragiles."

[Smolenski, 1992, p. 83]

Est-ce à dire que la déclarativité ne peut pas se passer de la logique ? Nous éluderons la question... La déclarativité ne fait pas partie de nos objectifs. En revanche, nous retiendrons de cette discussion l'importance, dans le processus de compréhension, de l'évaluation de l'ensemble des "utilisations possibles" d'une connaissance représentée...

La deuxième critique du paradigme déclaratif met en question l'indépendance proclamée des règles les unes par rapport aux autres. [Clancey, 1983] a fait remarquer que, malgré une façade d'uniformité dans la représentation des connaissances, les règles de MYCIN étaient définies de manière à guider le moteur d'inférence là où l'on voulait qu'il aille. Si le concepteur doit par exemple prêter attention à l'ordre des hypothèses des prémisses des règles, c'est bien le signe que les règles ne sont pas indépendantes.

"Quelquefois l'ordre des clauses compte avant toute chose (et l'ordre peut avoir différentes significations), et certaines règles sont présentes dans le seul but de contrôler l'invocation d'autres règles. L'uniformité de la représentation cache ces différentes fonctions des clauses et des règles"

[Clancey, 1983, p. 216]

L'exemple du rhume a aussi fait apparaître cette interdépendance entre les règles, qui met à mal l'argument de la compréhensibilité de "morceaux de connaissance" indépendants. Si la première critique attaquait déjà l'argument de la compréhensibilité d'une règle isolée, la seconde met aussi en cause la révisabilité des règles (du moins la facilité d'ajouter des règles dans la BC, qui est l'argument donné par les auteurs de MYCIN) :

"La modularité des règles elles-mêmes n'entraîne pas une modularité des conséquences des modifications apportées à ces règles"

[Gaines et Compton, 1995]

"Cependant, on se demande souvent si le fait de modifier le comportement des systèmes de production est aussi aisé que cela. Ajouter et retirer une règle a souvent des effets inattendus, de même que la modification des conditions d'une règle. Des modifications locales à la règle peuvent causer des modifications globales du comportement du système, comme dans n'importe quel programme"

[Jackson, 1990, p. 207]

La troisième critique porte sur la nature générale de l'objet "règle". Le paradigme déclaratif considère chaque règle comme universellement valable. Or, dans la réalisation pratique de SBCs, les conséquences de cette hypothèse peuvent être problématiques. Une explication à cela serait d'admettre la nature contextuelle de la connaissance ([Compton et Jansen, 1990]) et de reconnaître que la validité d'une connaissance est toujours à délimiter.

Ainsi, si l'on reste dans le cadre strict d'un ensemble de règles toutes au même niveau et indépendantes, la nature contextualisée de l'expertise risque de provoquer un "gonflement" des prémisses qui finit par remettre en question la lisibilité de la règle :

> *"Le format unique "SI.. ALORS..." entraîne une certaine lourdeur dans les expressions des membres gauches et une certaine répétition des mêmes prémisses pour des situations semblables. Il est ainsi malaisé d'exprimer des règles complexes"*
>
> [Laurière, 1986, p. 355]

Pour remédier à ce problème, il est possible d'utiliser un mécanisme efficace, qui a pour seul inconvénient de sortir du paradigme déclaratif initial[4]. L'idée est de regrouper les règles en différents *paquets* (qu'on peut appeler "contextes"), qui sont activés et désactivés par des règles, dites méta-règles. On distingue alors plusieurs niveaux de connaissances, puisqu'il faut au moins un niveau de **méta-connaissances** pour définir les activations et désactivations des paquets de règles. La lisibilité d'une règle isolée devient alors moins évidente, parce que son contexte de déclenchement n'est plus seulement défini par les prémisses de la règle, mais aussi par les méta-connaissances en mesure d'activer le paquet auquel appartient la règle.

En conclusion, l'idée d'attacher la connaissance à des *contextes d'utilisation*, représentés formellement, bat en brèche le concept de blocs de connaissances indépendants et isolément compréhensibles. Mais cette idée nous paraît très intéressante, parce qu'elle rejoint notre intuition de *localité* des connaissances : une connaissance n'est valable que dans un champ limité.

---

[4] Ces adaptations pragmatiques ont donné grandement satisfaction dans la pratique ; nous voudrions souligner simplement que les principes initiaux échafaudés autour de la notion de règle n'ont pas été concrètement respectés.

1.2.1.2 Les notions de tâche et de rôle

"Dans les représentations des connaissances traditionnelles, la connaissance était représentée indépendamment de la tâche"

[Chandrasekaran et Johnson, 1993, p. 237]

Lorsque l'on survole l'ensemble des travaux en acquisition des connaissances depuis une quinzaine d'années, on peut être frappé par l'émergence de deux notions qui étaient peu présentes ou ignorées dans les systèmes experts de première génération :

- la notion de méthode de résolution de problème, et plus généralement de **tâche**[5];
- et la notion de **rôle**.

L'apparition de ces deux notions est liée au basculement vers les systèmes experts dits de deuxième génération.

a) La notion de tâche

La première génération de systèmes experts ne s'intéressait pas à la modélisation explicite du "*comment (atteindre un but) ?*". Certes la notion de but, incontournable, était bien présente, mais ce qu'on appelle les connaissances de contrôle n'était par exemple pas exprimé explicitement et séparément. Le "comment ?" était en quelque sorte distillé dans la Base de Règles ou fixé dans l'algorithme du moteur d'inférence. [van Someren, Zheng et Post, 1990] introduisent les termes de paradigme "connaissances compilées" pour qualifier ce type de systèmes. Dans le paradigme "connaissances compilées", c'est le résultat donné par le système qui, seul, importe. Les concepteurs du système se soucient peu de détailler et d'expliciter *comment* la solution est trouvée.

"Parce que le système était conçu pour résoudre des problèmes, la plupart des connaissances qui n'étaient pas directement pertinentes pour trouver la solution était complètement abandonnées ou apparaissaient de manière implicite dans la formulation des connaissances au sein de la BC"

[van Someren, Zheng et Post, 1990, p. 340]

[5] Il est d'usage de distinguer le type de problème (appelé tâche) des moyens employés pour résoudre ce type de problème (la méthode). Nous ne faisons pas cette distinction. Nous allons revenir sur ce point très vite, mais pour l'heure lorsque nous employons le terme tâche, il faut comprendre qu'il s'agit d'une activité, donc de la combinaison d'un but ET d'une méthode.

En conséquence, et comme nous l'avons un peu évoqué dans la section précédente, cette première génération de systèmes experts a des difficultés pour faire évoluer son expertise et pour construire des explications de son comportement. [van Someren, Zheng et Post, 1990] opposent le paradigme "connaissances compilées" au paradigme "connaissances explicites", dans lequel s'inscrivent les systèmes experts de seconde génération.

A l'origine de la nouvelle génération de systèmes experts, des chercheurs comme Ronald Brachman et Allen Newell vont proposer de s'abstraire de l'influence des formalismes d'implémentation, pour raisonner à des niveaux supérieurs, comme le niveau épistémologique [Brachman, 1979], ou le niveau connaissances [Newell, 1982].

"Lorsque l'on utilise un cadre de description pendant l'étape d'analyse, on ne veut pas être biaisé vers un formalisme d'implémentation particulier"

[Wielenga et Breuker, 1986, p. 306]

Sous l'influence de cette idée d'une part, et d'autre part à la suite de travaux d'analyse de systèmes existants ([Clancey, 1985], [Chandrasekaran, 1986], [MacDermott, 1988],...), l'idée s'est imposée que l'expression des connaissances à un certain niveau conceptuel (ou à plusieurs) était nécessaire à l'acquisition des connaissances.

Ainsi, la méthodologie KADS propose de distinguer quatre niveaux conceptuels : domaine, inférence, tâche, stratégie ([Schreiber, Wielenga et Breuker, 1993]). Au niveau tâche, par exemple, on trouve :

"les connaissances sur 'comment les inférences élémentaires peuvent être combinées pour réaliser un certain but'"

[Schreiber, Wielenga et Breuker, 1993, p. 30]

Une autre approche distingue plusieurs niveaux conceptuels et parmi ceux-ci un niveau *tâche*. C'est l'approche componentielle de Luc Steels ([Steels, 1990], [Steels, 1993]). Steels définit un modèle de la tâche ("ce qui doit être accompli"), un modèle du domaine ("ce qu'on sait pour accomplir la tâche") et un modèle de la méthode ("comment utiliser les connaissances pour accomplir la tâche").

Une troisième approche est particulièrement intéressante, non pas spécifiquement à cause de son découpage en différents niveaux d'abstraction, mais parce qu'elle met véritablement la notion de tâche au centre du processus d'acquisition des connaissances : il s'agit de l'approche Tâche Générique, introduite par [Chandrasekaran, 1986]. Une Tâche Générique est à la fois un type de

problème, une représentation des connaissances et une stratégie d'inférence. La notion de Tâche Générique reflète l'idée que les connaissances exprimées dans un SBC sont fortement liées à leur "utilisation", c'est-à-dire aux tâches qu'elles permettent d'accomplir. Conjointement, la représentation des connaissances dépend de la stratégie d'inférence utilisée. On retrouve cette idée également dans l'approche Méthode à Limitation de Rôles développées par John MacDermott ([MacDermott, 1988]) :

"... il devrait être possible d'imaginer un ensemble de méthodes à limitation de rôles, où chaque méthode définit les rôles que doivent jouer les connaissances spécifiques à la tâche nécessaires, et les formes dans lesquelles ces connaissances doivent être représentées"

[MacDermott, 1988, p. 150]

D'une manière générale, les formalismes proposés pour exprimer le modèle de la tâche sont variables et dépendent beaucoup de ce que les auteurs mettent derrière la notion de tâche. Certains, par exemple, considèrent que la décomposition en sous-buts, représentée dans la structure de tâche, ne doit pas spécifier de contraintes sur l'ordre de réalisation de ces sous-buts ([Steels, 1993], [Chandrasekaran et Johnson, 1993]). Et inversement :

"Représenter le contrôle au niveau tâche fournit une représentation de la stratégie du système plus abstraite, qui pourra être pleinement exploitée dans l'explication du raisonnement"

[David, Krivine et Ricard, 1993]

Au-delà des divergences sur les notions de tâche, de méthode et de lien entre les deux, on peut retenir qu'il est devenu central en IA de représenter explicitement les "différentes étapes nécessaires à la résolution d'un problème". On a ainsi vu apparaître des modèles pour exprimer la décomposition en sous-buts, pour spécifier des ordres sur les opérations, pour représenter le contrôle sur ces opérations. C'est cette tendance générale que l'on veut souligner en parlant de "l'émergence de la notion de tâche".

Concluons cette section en justifiant pourquoi nous employons et insistons sur le terme tâche plutôt que celui de méthode. En premier lieu, il est difficile de concevoir la notion de tâche indépendamment des activités de la tâche : une tâche est plus qu'un but, c'est d'abord une activité. Nous ne sommes donc pas d'accord avec la définition de Mark Musen :

"Le terme tâche signifie la déclaration d'un problème d'application, sans se soucier de comment ce problème pourrait être résolu"

[Musen, 1989, p.353]

[Steels, 1990] distingue aussi très nettement la notion de tâche, qui définit le but à atteindre et les contraintes d'obtention de ce but, de la notion de méthode, qui décrit comment résoudre la tâche, soit en la décomposant en sous-tâches, soit en utilisant un ou plusieurs pas d'inférence élémentaires. Cette conception semble dominer dans le domaine de l'acquisition des connaissances.

Cependant, comme Joost Breuker nous pensons que la notion de *tâche* ne se réduit pas à la seule définition d'un but ou d'un problème :

"Cependant, par le terme tâche nous faisons référence à quelque chose qui peut être répété, peut être distribué et qui contient un plan, c'est-à-dire une configuration de méthodes de résolution de problème. Une tâche diffère d'un problème parce que dans une tâche, les méthodes de résolution de problème sont connues, alors que pour un problème, celles-ci doivent encore être trouvées"

[Breuker, 1994, p. 121]

Parallèlement, il est difficile de concevoir la notion de méthode, indépendamment d'une tâche précise :

"L'idée que la notion de méthode de résolution de problème a un sens indépendant de la notion de tâche a été défendue de manière convaincante par [Clancey, 1985], mais en pratique il semble que les méthodes restent étroitement liées aux tâches"

[Van de Velde, 1993, p. 220]

"Les méthodes sont en fin de compte toujours au service de buts. Les méthodes qui apparaissent être indépendantes de buts sont en fait soit des méthodes pour des problèmes très généraux, soit des méthodes pour des tâches très spécifiques, mais qui sont des sous-tâches de nombreuses tâches"

[Chandrasekaran et Johnson, 1993, p. 252]

De même que la connaissance en général, une méthode est liée à ses applications, à ses utilisations concrètes. On peut concevoir la notion de méthode comme étant à l'origine une tâche (opérationnelle), qui a été ensuite abstraite à des fins de réutilisation.

C'est donc bien à notre avis la notion de tâche qui est fondamentale (et à la source de la notion de méthode). Nous retiendrons le point de vue de l'approche CommonKADS ([Wielenga, Van de Velde, Schreiber et Akkermans, 1993]) qui définit la tâche comme "la suite des activités qui permettent d'atteindre un but".

b) La notion de rôle

Le second pas effectué, parallèle à la reconnaissance de la notion fondamentale de tâche, est un pas qui met à mal le paradigme déclaratif : il s'agit de reconnaître le lien étroit entre connaissances du domaine et tâche... C'est la notion de rôle qui va représenter ce lien, lorsqu'il est nécessaire de le représenter. La notion de rôle fait donc en quelque sorte son apparition avec l'idée que les connaissances représentées (typiquement les règles d'inférence) ne sont jamais vraiment indépendantes de leur utilisation pour l'accomplissement de tâches : elles leur sont liées parce qu'elles interviennent (elles jouent des "rôles") à chaque étape de la résolution.

Par exemple, Le modèle KADS cherche à maintenir une indépendance entre connaissances du domaine et tâches. C'est la couche inférence qui fait le lien entre le modèle du domaine et le modèle de la tâche. La notion de rôle apparaît donc de manière centrale, au sein la couche inférence, à la fois comme pointeur vers certaines connaissances du domaine et comme étiquette caractérisant la fonction des connaissances au sein de l'inférence.

L'approche opposée est représentée par l'école américaine, où n'apparaît pas la même représentation explicite de la notion de "rôle". Ainsi pour John MacDermott, une méthode à limitation de rôle est définie avec les connaissances de contrôle et les rôles qui doivent être remplis par les connaissances du domaine : l'acquisition des connaissances est alors toujours effectuée pour un rôle donné dans la résolution de problème. Mais un rôle est strictement attaché à sa méthode.

Dans les Tâches Génériques de B. Chandrasekaran n'apparaît pas du tout la notion de rôle, puisque les connaissances représentées sont locales et valables uniquement pour la tâche générique...

1.2.1.3 La notion de cas

"L'intuition du Raisonnement à Partir de Cas est que les situations surviennent avec régularité"

[Kolodner, 1993, p. 8]

La notion de cas est également une notion qui a émergé récemment en IA, sous deux formes distinctes. Si l'on prolonge le survol des travaux en acquisition des connaissances entamé dans la section précédente, on peut également retenir que la notion de **modèle du cas** s'est dégagée et a pris son importance, notamment au

sein des approches constructivistes d'acquisition des connaissances ([Van de Velde, 1993], [Causse, 1994]). Indépendamment, le **raisonnement à partir de cas** a pris un envol industriel en proposant de construire à moindre frais des Bases de Connaissances à partir de la simple description de "cas résolus".

a) Le modèle du cas

On peut trouver l'origine de la notion de modèle du cas dans le Modèle Spécifique du Patient (en anglais Patient Specific Model) du système de diagnostic médical ABEL ([Patil, Szolovits et Schwartz, 1981]). Le Modèle Spécifique du Patient dans ABEL contient toutes les données concernant le patient étudié, aussi bien que le réseau causal des hypothèses constituées à partir de ces données. Le processus de diagnostic consiste à construire et à réviser ce Modèle Spécifique du Patient. Celui-ci sert aussi de base pour fournir à l'utilisateur des explications du comportement du système.

William Clancey, à partir du système MYCIN, développe et généralise cette idée en insistant sur le fait que le modèle construit au cours de la résolution, qu'il appelle Modèle Spécifique de la Situation (MSS ou en anglais "Situation Specific Model") décrit un processus :

"... le réseau qui relie les symptômes et les maladies est un modèle de séquences particulières d'événements dans le monde (appelé aussi modèle spécifique de la situation)"

[Clancey, 1986, p. 55]

"En résumé, un MSS est un modèle des processus qui se déroulent dans un système"

[Clancey, 1992, p. 33]

Le MSS possède les mêmes caractéristiques que le Modèle Spécifique du Patient d'ABEL :

- c'est une **preuve**, un argument causal,

"C'est l'idée qu'un diagnostic n'est pas le nom d'une maladie mais un argument qui relie de manière causale les manifestations qu'il faut expliquer (parce qu'elles sont anormales) aux processus qui les ont amenées"

[Clancey, 1986, p. 55]

- il a un grand intérêt par sa **capacité explicative** : c'est le graphe du MSS qui doit permettre à l'utilisateur de comprendre le raisonnement du système. Cette idée est également présente dans le système ROGET ([Bennett, 1985]), où un

éditeur graphique présente à l'utilisateur une "structure conceptuelle" sous forme de graphe direct acyclique.

- il **guide** la résolution de problème, d'une manière qui semble inspirée du modèle "tableau noir" : le MSS est une structure unique, centrale, qui résume complètement l'état de la résolution du problème, et qui est enrichie par les inférences effectuées par le système. Pour cette raison, le MSS est une représentation d'un processus et il contient toute l'histoire du raisonnement.

"L'idée clé de notre argumentation est que c'est cette description interne, le MSS, est inspectée par le programme lui-même durant le raisonnement, et son état partiel, vu de manière structurelle en termes globaux (par opposition à la recherche uniquement d'assertions spécifiques) est utilisé pour conduire les opérations d'inférence"

[Clancey, 1992, p. 26]

Dans le prolongement de cette idée, Luc Steels appelle **modèle du cas** la situation de résolution de problème ([Steels, 1990]). Un modèle du cas est modifié au fur et à mesure de la résolution du problème. C'est ce qui le différencie, dans l'approche componentielle, d'un *modèle du domaine*, qui représente une connaissance figée durant toute la résolution du problème.

"Le résolveur de problème utilise les modèles du domaine pour étendre le modèle du cas par inférence ou accumulation de données"

[Steels, 1990, p. 29]

Modèles du domaine et modèles du cas jouent tous des *rôles particuliers* dans le raisonnement. Dans l'approche componentielle, des diagrammes de dépendance représentent ces modèles et leur rôle sous forme d'un flux d'informations. Mais comme le remarque Karine Causse ([Causse, 1994, p. 137]) la notion globale de Modèle de la Situation Spécifique est ici éparpillée au sein d'un ensemble de modèles du cas successifs, dont les liens respectifs ne sont pas très précisément représentés. Karine Causse introduit la notion de Modèle Conceptuel du Cas (MCC) qui se distingue du Modèle de la Situation Spécifique par le fait que c'est un objet qui représente *l'interprétation* du problème par un concepteur ou cogniticien. Cette interprétation n'est a priori pas aussi figée que l'est le MSS de William Clancey. Un schéma de MCC, exprimé via un langage de primitives de représentation, est un modèle plus général, plus souple, plus riche qu'un MSS.

Walter Van de Velde résume ce type d'approche "constructiviste" de l'acquisition des connaissances, qu'il appelle résolution de problème vue comme

modélisation ("problem solving as modelling"), de la manière suivante ([Van de Velde, 1993]):

"La résolution de problème est donc vue comme la 'création' d'un modèle du cas adéquat et l'interaction avec le monde est seulement une ressource pour cela, presque un effet de bord dans le processus de maintenir une organisation interne et une identité"

[Van de Velde, 1993, p. 223]

Dans cette approche constructiviste, les actions sont les moyens d'obtenir des informations sur le monde, non le but de la résolution de problème.

Le modèle du cas est une instanciation des hypothèses développées aux niveaux domaine, tâche et méthode, à lumière des données connues sur le cas ("case data") :

"La résolution de problème comme modélisation est un processus d'instanciations d'hypothèses (domaine et tâche) reliées par la théorie de la 'compétence de la méthode' et utilisant les données du cas supposées à partir de l'interaction avec le monde"

[Van de Velde, 1993, p. 222]

Le modèle du cas prend donc un rôle central dans la résolution de problème vue comme modélisation :

"Les systèmes experts de la prochaine génération s'appuient sur le processus de résolution de problème comme modélisation, en rendant explicite les modèles du cas qu'ils construisent, fondant toute interaction avec le monde sur le besoin de faire évoluer ce modèle du cas vers un état comprenant certaines caractéristiques"

[Van de Velde, 1993, p. 224]

En conclusion, on peut retenir que la notion de modèle du cas est une notion dynamique : un modèle qui évolue au fur et à mesure de la résolution de problème, qui s'instancie même, à la manière d'un schéma[6] (au sens de [Richard, 1990]).

---

[6] Les approches constructivistes en acquisition des connaissances s'inspirent d'ailleurs de la psychologie cognitive (par exemple [Causse, 1994], [Cañamero, 1995]). Cette idée d'instanciation peut nous donner l'occasion de revenir sur la fin de la citation précédente de W. Van de Velde : est-il vraiment nécessaire d'expliciter à l'avance les caractéristiques du modèle du cas désiré ? Autrement dit, le système fait-il évoluer son modèle du cas dans le but explicite de satisfaire certaines caractéristiques ou le système se laisse-t-il vraiment complètement guider par le modèle du cas courant ? La seconde réponse est davantage cohérente avec l'approche constructiviste, mais s'oppose à la vision classique de la résolution de problème, qui repose sur la notion de but. Or le processus d'instanciation de schémas reliés hiérarchiquement montre qu'il tout à fait possible de partir d'un modèle initial pour parvenir finalement à un modèle final (appartenant à un ensemble de "modèles buts", liés au modèle initial par la hiérarchie) sans jamais exprimer aucune autre contrainte que le modèle initial lui-même. Nous reviendrons sur ce point lorsque nous présenterons la notion d'*attente* dans la section 1.3.4.

Cette vision s'oppose à celle du Raisonnement à Partir de Cas pour laquelle le cas est vu comme un objet statique (une description d'un problème résolu), qu'on peut mémoriser avec l'objectif de le retrouver lorsqu'il pourra être utile pour le problème courant. Cependant les deux approches ont en commun de mettre la notion de cas au centre du raisonnement.

b) Le Raisonnement à Partir de Cas

"Le RPC est aussi une approche pour l'apprentissage incrémental soutenu, puisqu'une nouvelle expérience est retenue à chaque fois qu'un problème a été résolu, rendant celle-ci immédiatement disponible pour les futurs problèmes"

[Aamodt et Plaza, 1994, p. 39]

Les principes du Raisonnement à Partir de Cas (Case-Based Reasoning ou CBR en anglais) reposent sur la notion de cas, vu comme la description d'une expérience passée, et s'appuient sur une certaine vision de ce qu'est l'intelligence et la compréhension, initiée en particulier par Roger Schank :

"De fait, à chaque fois qu'il y a un cas antérieur disponible pour fonder le raisonnement, les gens vont le retrouver et l'utiliser comme modèle pour leur future prise de décision"

[Riesbeck et Schank, 1989, p. 6]

"Nous vivons dans un monde de cas. Notre compétence à utiliser ces cas de nouvelles manières est l'empreinte de ce que l'on appelle l'intelligence"

[Riesbeck et Schank, 1989, p. 389]

"Comprendre veut dire se remettre en mémoire l'expérience vécue la plus proche[7]"

[Riesbeck et Schank, 1989, p. 18]

Le Raisonnement à Partir de Cas (RPC) consiste schématiquement à indexer une bibliothèque de cas préétablis, définis chacun par un ensemble de caractéristiques, puis à apparier les caractéristiques du cas à traiter avec celles des cas déjà connus. Le système, par des techniques basées sur des mesures de similarité, identifie alors un cas "similaire" au cas à traiter, pour lequel une solution est connue. Une étape d'adaptation du cas remémoré permet de proposer une solution pertinente pour le cas à traiter.

Le RPC présente plusieurs avantages par rapport aux systèmes à base de règles. En premier lieu, le RPC apporte plus de **flexibilité** au raisonnement,

[7] "the closest previously experienced phenomenon"

puisque le principe du RPC est de construire des solutions pour des cas qui n'ont jamais été rencontrés auparavant par le système. Or un système à base de règles est conçu pour fonctionner pour un ensemble de situations bien délimité à l'avance :

"Il [le système à base de règles] couvre le normal, il ne nous dit rien sur comment raisonner quand les situations diffèrent de la norme"

[Kolodner, 1993, p. 8]

Dans le prolongement de cette idée, le RPC est une réponse à notre objectif de construction de **systèmes évolutifs**. En effet, l'avantage des systèmes flexibles est de pouvoir fonctionner lorsque les connaissances du domaine ne sont que partiellement identifiées :

"Le RPC est aussi utile quand la connaissance est incomplète et/ou très peu évidente"

[Kolodner, 1993, p. 24]

"Les cas sont utiles pour interpréter les concepts sans limites fixes et mal définis"

[Kolodner, 1993, p. 26]

Lié à tout ceci, l'énorme avantage du RPC est de permettre de considérablement **diminuer le travail d'analyse** du domaine :

"... beaucoup de domaines du monde réel sont si complexes qu'il est impossible ou irréalisable de spécifier complètement toutes les règles qui entrent en jeu; par contre les cas, c'est-à-dire les solutions aux problèmes, peuvent toujours être donnés"

[Riesbeck et Schank, 1989, p. 26]

Le revers de la médaille est dans le problème de la prescriptivité. Le RPC n'est pas bien adapté au raisonnement normatif : une norme et ses exceptions sont représentées de la même manière (ce sont des cas). La souplesse intrinsèque du RPC fait que le risque est toujours présent de remémorer une exception plutôt que la norme :

"Un système de RPC peut permettre aux cas de trop le biaiser dans la résolution d'un nouveau problème"

[Kolodner, 1993, p. 26]

D'autre part, la représentation à partir de cas ne cherche pas à construire de raisonnement explicite menant à la solution : le processus de remémoration est difficile à décomposer et à expliquer à l'utilisateur... En cela, le RPC peut-être vu

comme un modèle type "boîte noire" et ne correspond pas aux objectifs que nous nous sommes fixés. Toutefois, certains systèmes de RPC comme CYRUS ([Kolodner, 1993]) sont organisés hiérarchiquement en "réseaux de discrimination redondants" : les "caractéristiques" des cas servent à regrouper les cas et à identifier des types de situations. Les caractéristiques jouent alors un double rôle :

"... nous parlons des caractéristiques comme décrivant les situations dans lesquelles les cas peuvent fournir des directives pour le futur, et nous pensons aux caractéristiques comme remplissant la fonction de distinguer conceptuellement les cas entre eux"

[Kolodner, 1993, p. 383]

Dans ces conditions, le processus de remémoration peut très bien être décomposé et rendu moins opaque à l'utilisateur.

En conclusion, la notion de cas dans le RPC est très intéressante. C'est un objet **explicite** qui répond à nos exigences de **précision** et de **localisation** des connaissances. Un système de RPC est censé stocker chaque expérience exactement telle qu'elle a eu "lieu", sans transformation[8]. Chaque cas présent dans un système de RPC témoigne d'une expérience précise, enracinée dans la réalité. De plus, un cas est toujours exprimé à un **niveau opérationnel**, puisque c'est un compte-rendu d'une expérience :

"Les cas, qui représentent des connaissances spécifiques attachées à des situations spécifiques, représentent les connaissances à un niveau opérationnel; effectivement ils rendent explicite comment une tâche a été accomplie ou comment un morceau de connaissance a été appliqué ou quelle stratégie particulière a été utilisée pour remplir un but"

[Kolodner, 1993, p. 9]

La notion de **cas abstrait** est une perspective intéressante du RPC qu'ouvre Janet Kolodner :

"... je dois admettre cependant que la plupart de nos systèmes de RPC implémentés utilisent très peu de descriptions abstraites ou générales de situations, et s'appuient presque complètement sur des cas pour les descriptions"

[Kolodner, 1993, p. 11]

Développer cette notion de cas abstrait apparaît comme une direction de recherche prometteuse pour obtenir une représentation plus explicite des raisonnements.

---

[8] Il est vrai qu'il s'agit toujours d'une interprétation...

1.2.1.4 La notion d'activation

La notion de propagation ou d'activation est très souvent présente dans les modèles inspirés des sciences cognitives ou visant à simuler l'activité cognitive. La propagation de l'activation est une idée qui provient de la métaphore neurologique. Elle consiste à considérer qu'une structure devient active dès lors qu'elle a été déclenchée (activée) par une autre, celle-ci pouvant alors se désactiver ou non. Une structure activée procède à un certain nombre d'actions, qui peuvent elles-mêmes consister en l'activation d'autres structures.

Le modèle **connexionniste** est le premier modèle fondé sur la notion d'activation qui vient à l'esprit. Le mouvement connexionniste en IA s'est détaché des représentations explicites du raisonnement pour s'intéresser à la construction de "réseaux de neurones".

"Dans ces modèles [les modèles connexionnistes], le traitement se fait par la propagation de l'activation au sein d'un grand nombre d'unités de traitement simples"

[MacClelland, 1991, p. 41]

Les réseaux de neurones sont des systèmes capables d'apprendre à reconnaître des "entrées". Ils s'appuient sur des mécanismes d'activation et d'inhibition de structures appelées neurones conformément à la métaphore. Un neurone formel est caractérisé par trois paramètres ([Ganascia, 1990]) : un état interne, un voisinage de neurones et une fonction de transition. L'état interne des neurones de la première couche se propage dans le réseau jusqu'à la dernière couche qui détermine les "sorties".

On retrouve aussi la notion d'activation dans des modèles davantage explicites, comme le modèle d'**acteur**, proposé à l'origine par Carl Hewitt ([Hewitt, Bishop et Steiger, 1973]). Comme les réseaux de neurones, la connaissance est ici complètement répartie à travers l'ensemble des structures, et il n'y aucune "connaissance globale" (type "tableau noir"). Chaque objet, appelé acteur est un agent actif, autonome, communiquant librement (sans contrôle centralisé) avec ses semblables. Le comportement d'un acteur est défini par un script, unique, qui filtre les messages reçus et active la partie appropriée du comportement. L'activation d'un autre acteur est faite par envoi de messages :

"Notre formalisme montre comment tous les modes de comportement peuvent être définis exclusivement par un seul type de comportement : l'envoi de messages à des acteurs"

[Hewitt, Bishop et Steiger, 1973, p. 235]

"Lorsqu'un acteur a accès à une nouvelle information, il la propage aux acteurs qu'il connaît, qui, à leur tour, peuvent la propager et éventuellement en déduire de nouvelles informations. La connaissance est diffusée parmi l'ensemble des acteurs"

[Masini, Napoli, Colnet, Léonard et Tombre, 1989, p. 375]

Un autre modèle explicite où la notion d'activation est présente est le modèle de "**frame**" (cadre), introduit par [Minsky, 1981] pour représenter explicitement des modèles de situations.

"Les frames sont un formalisme de représentation créé pour prendre en compte des connaissances qui se décrivent mal en calcul de prédicats ; typicalité, valeurs par défaut, exceptions, informations incomplètes ou redondantes"

[Masini, Napoli, Colnet, Léonard et Tombre, 1989, p. 286]

Marvin Minsky compare un frame à un formulaire comportant des cases à remplir. Le remplissage de certaines cases peut nécessiter l'activation d'autres frames (déclenchement de "réflexes"). L'inconvénient des systèmes à base de frames est que la dynamique générale du déclenchement des réflexes n'est pas facilement lisible et peut difficilement être contrôlée. Le modèle de frame recherche un certain réalisme psychologique et il n'a pas pour objectif de permettre de prescrire des savoir-faire : ces deux objectifs sont à notre avis bien distincts.

La notion d'activation est un concept central également dans les sciences cognitives. Comme le cerveau procède par activations, les modèles cognitifs proposés sont souvent fondés sur des structures activées ou "activantes". Citons par exemple le modèle des "**p-prims**" (*phenomenological primitives*) d'Andy diSessa ([diSessa, 1987], [diSessa, 1993]). Les p-prims sont les structures mentales les plus atomiques qu'on puisse isoler, des abstractions simples, ni concepts, ni éléments purement sensoriels. Un p-prim est une "évidence", il ne nécessite aucune explication (exemple de p-prim : "maman-est-là"). Ces structures sont capables d'activer et de désactiver les autres structures, prioritairement les structures les plus "voisines".

"L'apprentissage devrait faire en sorte que les p-prims soient activés dans des circonstances appropriées, et à leur tour ils devraient contribuer à activer d'autres éléments en fonction des contextes qu'ils spécifient"

[diSessa, 1993]

Le modèle des p-prims étant un modèle psychologique, il n'a aucun caractère prescriptif : l'activation des p-prims est faite par un mécanisme similaire au modèle connexionniste ([diSessa, 1993]).

En conclusion, nous retiendrons de la notion d'activation qu'elle peut être intéressante pour nous lorsqu'elle est utilisée dans un contexte séquentiel parce qu'alors dans ce contexte :

- les connaissances sont localisées dans des structures, donc circonscrites à un "lieu" identifiable et "inspectable" (par exemple un frame),

- les activations explicites de ces structures fournissent une "traçabilité" du raisonnement, caractéristique importante pour remplir notre objectif de prescriptivité.

1.2.2 Des travaux et des domaines de recherche

Poursuivons notre panorama des courants d'idées qui touchent à notre objectif de recherche et proposent des solutions intéressantes. Après les concepts-clé en IA (règles, tâches et rôles, cas, activation), penchons-nous maintenant sur trois types de travaux en IA : la programmation, orientée objet, les systèmes apprentis, et la cognition située.

1.2.2.1 Les environnements de programmation orientée objet

Nous associons dans cette section les langages orientés objet à leurs environnements de développement :

"Il n'est plus raisonnable de concevoir un langage dissocié de son environnement de programmation"

[Masini, Napoli, Colnet, Léonard et Tombre, 1989, p. 15]

Si la notion d'objet a provoqué une révolution dans le domaine de la programmation, cette révolution n'aurait sans doute pas eu lieu sans les techniques d'interface personne-machine qui ont rendu possible et conviviale la manipulation des objets dans les environnements de programmation. Ainsi le multifenêtrage apparaît essentiel pour permettre la visualisation des hiérarchies de classes au moyen de "browsers". Les menus contextuels sont également indispensables pour proposer au développeur des opérations adaptées au contenu des fenêtres affichées. Les environnements de programmation orientée objet sont donc difficilement séparables du langage qu'ils soutiennent, ce d'autant plus lorsque le langage est

interprété et qu'alors l'environnement intègre l'interprète du langage (par exemple, Smalltalk ou les Lisp dotés d'une composante objet).

Pour présenter très vite ce qu'a été la révolution de la programmation objet, on peut dire qu'il s'agit du passage d'une programmation dirigée par les traitements à une programmation dirigée par les données. Avec la notion d'objet, les données deviennent centrales, elles sont organisées en hiérarchie de classes et les procédures leur sont attachées.

Compréhensibilité et révisabilité font partie des arguments en faveur de la programmation orientée objet. Une des principales caractéristiques de la modélisation "objet", la modularité, permet d'une part la **modifiabilité**, puisqu'il devient facile de changer la définition d'un objet avec un minimum d'interaction sur les autres objets, et d'autre part la **lisibilité** avec la notion d'encapsulation qui fait en sorte de cacher les détails d'implémentation de l'objet en montrant uniquement *l'interface* de l'objet, c'est-à-dire l'ensemble des opérations applicables à l'objet.

Le mécanisme d'héritage permet de définir des sous-classes par différenciation et permet une plus grande concision des programmes qui joue aussi en faveur de la compréhensibilité des programmes.

La notion d'objet présente donc des qualités qui sont indéniablement proches de nos objectifs. Elle en apporte même d'autres, comme la **réutilisabilité** : l'objet étant défini par une interface explicite, il doit pouvoir être facilement réutilisé dans d'autres contextes que celui pour lequel il a été programmé initialement.

Si modularité et mécanisme d'héritage sont classiquement les caractéristiques citées du paradigme objet, nous voudrions y ajouter avec [Ganascia, 1990] la notion d'**identificateur d'objets** qui nous parait également centrale. C'est un autre argument pour ne pas dissocier *environnement* de programmation objet et *langage* de programmation objet. En effet, la programmation par objet perdrait sans doute tout intérêt sans l'indexation des objets qui permet l'accès instantané au contenu d'un objet à partir de son seul nom. Ainsi, l'image virtuelle de Smalltalk contient tous les objets prédéfinis et les objets propres aux applications développées. La persistance des objets dans l'environnement de développement facilite énormément la programmation et le débogage en offrant par exemple un accès direct à tous les objets qui font référence à un objet donné o, ou encore l'accès à toutes les méthodes faisant appel à une méthode m[9]...

On retiendra finalement des environnements de programmation orientée objet :

---

[9] Comme "tout est objet" dans l'environnement Smalltalk, une méthode est aussi un objet et cette opération qui consiste à chercher tous les contextes dans lesquels une méthode m1 peut être utilisée, se rapporte à une recherche de référents d'instances.

- la notion d'objet, unité sémantique encapsulée, lisible, indexable, dont les éventuelles optimisations internes ne nécessitent pas de changements externes,

- l'importance des facilités de navigation dans les bases d'instances et bases de classes,

- l'intérêt des présentations graphiques de ces structures (pour les hiérarchies de classes).

La correction des erreurs dans leur contexte d'exécution est une autre caractéristique intéressante des environnements de programmation orientés objet, mais elle ne leur est pas propre. C'est en effet une caractéristique qu'on trouve à l'origine dans l'environnement de programmation du langage LISP, comme le rappellent [Masini, Napoli, Colnet, Léonard et Tombre, 1989]. La réduction du cycle édition - compilation - édition de liens - test dans les environnements de développement à partir de langages interprétés, rend possible ce type de débogage interactif, ainsi qu'une "programmation incrémentale".

Apprendre / corriger dans le contexte du programme en exécution, "en situation", est une idée maintenant classique en programmation. Les outils de débogage, qui s'attachent à expliciter le contexte de l'interruption du programme ainsi que son historique pour aider à corriger le programme, sont couramment utilisés.

Cette idée trouve un prolongement dans le domaine de l'apprentissage automatique avec la notion de système apprenti.

1.2.2.2 Les systèmes apprentis

Un système apprenti est défini par le couplage entre un système expert exploitant une BC, et un système d'apprentissage capable d'enrichir automatiquement la BC "au vol", lorsque l'utilisateur, après avoir détecté une défaillance du système expert, donne au système d'apprentissage une solution concrète au problème courant. Cette idée a été formulée au départ par Tom Mitchell :

"... nous devrions trouver des moyens de permettre à ces systèmes [les systèmes experts] d'apprendre à travers leurs interactions normales avec leurs utilisateurs"

[Mitchell, 1983b, p. 1139]

"Un Système Apprenti est un assistant interactif pour construire et mettre au point une Base de Connaissances. Ses buts sont doubles : (1) automatiser partiellement la construction initiale d'une BC en générant automatiquement des

règles de surface à partir d'une théorie du domaine approximative ; (2) interagir avec les utilisateurs pour aider au raffinement des connaissances à travers l'expérience accumulée pendant la résolution de problème normale"

[Smith, Winston, Mitchell et Buchanan, 1985, p. 573]

Cette idée "d'apprendre par l'action" ([Kodratoff, 1986]) a donné naissance à un certain nombre de systèmes apprentis comme LEX ([Mitchell, Utgoff et Banerji, 1983]), LEAP ([Mitchell, Mahadevan et Steinberg, 1990]), ODYSSEUS ([Wilkins, 1990]), DISCIPLE ([Tecuci et Kodratoff, 1990]), PROTOS ([Bareiss, Porter et Wier, 1990]), APT ([Nédellec, 1991]) et CAP ([Dent, Boticario, MacDermott, Mitchell et Zabowski, 1992]).

L'ancêtre des systèmes apprentis est sans doute le système TEIRESIAS ([Davis, 1979]), qui tirait déjà parti des avantages de ce qu'on peut appeler "l'apprentissage en situation". Les avantages de cette démarche sont les suivants :

- la communication entre utilisateur et système apprenti est facilitée puisque c'est un cas **concret** (correspondant à une exécution précise du système) qui est à l'origine de l'apprentissage.

"L'intégration de l'acquisition des connaissances et des techniques d'apprentissage permet au système [apprenti] d'apprendre correctement comment résoudre les problèmes à partir d'exemples concrets réels"

[Nédellec et Causse, 1992, p. 172]

Plus généralement, cet aspect "concret" est un aspect important dans l'apprentissage, que Roger Schank appelle le "critère du contexte" : avoir un contexte disponible pour placer la nouvelle information ([Schank, 1982b]).

- l'apprentissage s'appuie sur **un seul** cas concret et s'y restreint : seule une portion de la BC sera corrigée. Cette caractéristique est utilisée également par les techniques d'apprentissage à partir d'explications (Explanation-Based-Learning, ou EBL [Dejong et Mooney, 1986]). L'EBL construit une "explication" de l'exemple donné et généralise cette structure pour fournir une définition opérationnelle du concept auquel appartient l'exemple. [Mitchell, Keller et Kedar-Cadelli, 1986] opposent l'apprentissage basé sur les explications à l'apprentissage par similarité. Tandis que le premier part d'**une théorie** du domaine **et un exemple** concret, le second fonctionne à partir d'**un ensemble d'exemples** concrets. L'apprentissage à partir d'explications est une technique qui permet un apprentissage incrémental, mais dont l'objectif est d'améliorer les performances du système, et non de permettre de corriger la BC.

Une composante importante des systèmes apprentis, mentionnée d'ailleurs dans la définition de [Smith, Winston, Mitchell et Buchanan, 1985] donnée ci-dessus, est leur interactivité avec l'utilisateur au cours de l'apprentissage.

Ainsi, pour [De Raedt et Bruynooghe, 1992], les deux principales caractéristiques d'un système apprenti sont d'être convivial (“user-friendly”) et opportuniste :

“Un système est amical avec l'utilisateur s'il peut être utilisé de manière simple par des utilisateurs qui ne sont pas familiers avec les techniques sous-jacentes, et opportuniste s'il saisit les occasions pour apprendre à chaque fois qu'une possibilité se présente”

[De Raedt et Bruynooghe, 1992, p. 108]

Certains concepteurs de systèmes apprentis cherchent à limiter, voire à supprimer, cette interaction après le moment où l'utilisateur a détecté et fourni au système un exemple à apprendre (par exemple [Mitchell, Mahadevan et Steinberg, 1990]). Les problèmes qui se posent alors relèvent d'un autre domaine, celui de la conception de systèmes assistants intelligents ou de l'apprentissage par démonstration :

“La programmation par démonstration s'intéresse à l'apprentissage de procédures par généralisation des actions enregistrées de l'utilisateur”

[Lieberman, 1996, p. 62]

En résumé, le concept général de système apprenti est un moyen de remplir notre objectif d’incrémentalité. A l'instar des systèmes NeoDISCIPLE ([Tecuci, 1992]) et APT, il est même possible d'intégrer à un système apprenti des techniques de révision et d'obtenir ainsi des systèmes apprentis capables de corriger leur BC. Le système d'apprentissage intègre alors des compétences qui lui permettent, de manière plus ou moins automatique :

- de repérer une erreur,
- de localiser précisément l'origine de l'erreur,
- de corriger la BC de manière adéquate,
- de généraliser la correction.

C’est une véritable **coopération** qui se met alors en place entre un utilisateur / concepteur de la BC et le système. Chacun exerce une compétence complémentaire à l’autre partenaire. Le système est doué de capacités de calcul et de survol de la totalité de la BC, le concepteur possède (ou a les moyens de connaître) la connaissance à représenter.

Une particularité des systèmes apprentis, essentielle à nos yeux, est le fait qu'ils fonctionnent "en situation" : la révision démarre à partir de la détection "au vol" d'une défaillance du système. Cette idée rejoint quelque part le mouvement dit "cognition située", dont un aspect concerne l'étude de l'activité "en situation", et qui plus généralement met l'accent sur la pratique, par opposition à la théorie ([Lave, 1988]).

1.2.3 La cognition située

Nous terminons ce chapitre sur l'héritage par une présentation du mouvement dit action située ou cognition située. Ce mouvement d'idées s'est développé dans les sciences cognitives au cours des années 80 et 90, et le questionnement critique qu'il apporte à l'IA est d'autant plus intéressant qu'il appuie la démarche que nous nous sommes fixée dans la section 1.1 (comment permettre la construction incrémentale d'une BC, révisable à tout moment). Nous commençons par présenter les idées défendues, principalement par William Clancey, puis nous développons notre propre interprétation des problèmes soulevés par la cognition située dans le cadre qui nous concerne. C'est ainsi que nous en viendrons à définir et à défendre un axe de recherche, qu'on appellera "révision située".

1.2.3.1 Présentation

Le mouvement *action située* ou *cognition située* est un mouvement typiquement interdisciplinaire. On peut chercher son origine en ethnologie, sociologie, psychologie, philosophie, dans les sciences de l'éducation, mais aussi en robotique (ce que l'on appelle la *robotique située* : [Agre et Chapman, 1987], [Brooks, 1991], [Cohen, 1988], [Kaelbling, 1986]). Nous axerons notre présentation de la cognition située sur [Winograd et Flores, 1986], [Lave, 1988], et bien sûr des travaux de William Clancey ([Clancey, 1991a], [Clancey, 1991b], [Clancey, 1993a], [Clancey, 1993b], [Clancey, 1997a], [Clancey, 1997b]).

Dans [Winograd et Flores, 1986], Terry Winograd et Fernando Flores critiquent l'approche traditionnelle rationaliste, sur laquelle l'IA s'est développée, et qui impose sa vue de ce qu'est l'intelligence, la raison, la compétence.

"Non seulement l'orientation rationaliste soutient la science pure et appliquée, mais elle est également considérée, peut-être en raison de son prestige et du succès de la science moderne, comme le vrai paradigme de ce que veut dire penser et être intelligent"

[Winograd et Flores, 1986, p. 42]

Les auteurs s'appuient sur l'herméneutique et la phénoménologie, qui :

"privilégient les domaines de l'expérience humaine où l'interprétation individuelle et la compréhension intuitive (en tant qu'elles diffèrent de la déduction logique et la réflexion consciente) jouent un rôle prépondérant"

[Winograd et Flores, 1986, p. 33]

T. Winograd et F. Flores développent l'idée que la rationalité n'existe pas sans la *responsabilité* et qu'une différence fondamentale entre humains et ordinateurs apparaît dans le fait que les ordinateurs ne peuvent pas *s'engager*.

L'ethnologue Jean Lave rejoint la critique de l'approche traditionnelle rationaliste (qu'elle appelle cognitiviste ou fonctionnaliste) de [Winograd et Flores, 1986]. Pour J. Lave, plusieurs aspects de l'activité humaine et de la connaissance sont ignorés par la tradition cognitiviste :

- la nature **socialement** située de l'activité.

"Les caractéristiques centrales [de la tradition fonctionnaliste] comprennent la séparation de la cognition et du monde social, la séparation de la forme et du contenu mise en oeuvre dans la pratique de l'investigation isomorphique de la résolution de problème, et une explication strictement cognitive de la continuité dans l'activité à travers les situations. Tout cela dissocie la cognition de ses contextes, et participe à l'absence de théorisations sur les expériences comme situations sociales et la cognition comme activité socialement située"

[Lave, 1988, p. 43]

- la nature pratique, concrète et **incarnée** de l'activité.

"Il est vraiment très difficile de faire des théories sur les contextes, parce que les traditions théoriques les plus pertinentes ne prennent pas l'expérience dans le monde réel comme des objets d'analyse. Elles ont tendance à ignorer la nature incarnée, inéluctablement "localisée" de l'activité dans l'espace-temps, peut-être parce que c'est là quelque chose d'incohérent avec d'autres hypothèses"

[Lave, 1988, p. 148]

- la nature **ouverte** et contingente de l'activité.

"En résumé, les transformations de l'activité ne forment pas un ensemble fermé de possibilités logiques, mais elles sont ouvertes et contingentes"

[Lave, 1988, p. 189]

William Clancey s'est fait le porte-parole en IA du mouvement cognition située, en allant s'inspirer notamment des travaux du psychologue Frédéric Bartlett

ou du philosophe John Dewey[10]. Lui-même acteur important de la recherche en IA (cf. section 1.2.1), W. Clancey a le mérite de confronter sa propre discipline aux critiques issues du mouvement cognition située, et à chercher à en tirer les conséquences pour revisiter les concepts de l'IA (qu'est-ce qu'une connaissance, un symbole, etc.).

Nous retenons principalement deux points développées par W. Clancey, que nous reformulons à notre manière :

- La représentation n'est pas la connaissance,
- Il existe une "intelligence en situation" que l'IA ne reproduit pas.

a) une représentation reste toujours une représentation : il faut être attentif à ne pas assimiler la représentation à la connaissance qu'on cherche justement à exprimer par cette représentation.

"La connaissance ne peut pas être approchée facilement, elle peut seulement être pensée comme le résultat de processus interprétatifs qui opèrent sur des expressions symboliques"

[Newell, 1982, p. 105]

"La rupture de Newell avec l'approche conventionnelle en IA est de dire que les représentations des connaissances sont les jugements que porte l'observateur dans la théorie du Niveau Connaissances. Ils représentent les connaissances attribuées à l'agent, elles ne sont pas identifiées aux connaissances elles-mêmes de l'agent : la carte n'est pas le territoire"

[Clancey, 1991a, p. 381]

Il y a, derrière la question purement terminologique, l'idée que le résultat du processus de modélisation et de représentation (la BC) est par nature restreint, limité, incomplet, comparativement à la compétence de l'expert.

"Les formes adaptées du comportement intelligent produit par l'interaction entre les processus neuronaux et environnementaux (par exemple les stratégies, les habitudes, les sens des mots, les conventions de discours) — décrites par un observateur situé dans le temps et avec un certain cadre de référence — ne peuvent être attribuées à quelque chose préexistant en interne dans les structures neuronales (par exemples des scripts, des schémas, des règles) ou préexistant dans le monde (par exemple des propriétés, des objets ou des événements)."

[Clancey, 1991b, p. 110]

"Notre proposition est que le modèle est simplement une abstraction, une description et un générateur de motifs de comportement dans le temps, et non un mécanisme équivalent à la capacité humaine"

[Clancey, 1993a, p. 88]

[10] On peut citer également Bateson, Edelman, Gibson ou Maturana, comme sources de réflexions et d'influences citées par W. Clancey.

[Sandberg et Wielenga, 1991] ont fait remarquer que peu de chercheurs en IA font réellement cette "hypothèse d'identité" entre représentations informatiques et réelles représentations mentales. Pourtant, la controverse éclate (voir par exemple [Vera et Simon, 1993]) lorsque Clancey défend que fondamentalement, la pratique ne se réduit pas à la théorie ([Clancey, 1993a, p. 93]) :

"La connaissance ne peut être réduite à (pleinement saisie par) un corps de représentations. La connaissance ne peut être inventoriée"

[Clancey, 1993b, p. 37]

Conjointement ou conséquemment à cette réflexion générale sur la nature de la connaissance, le modèle construit (la BC) reste toujours le fait d'un (ou plusieurs) concepteur(s), et garde un statut subjectif.

"Les descriptions au Niveau Connaissances portent sur des agents, mais elles appartiennent au théoricien. Elles constituent sa connaissance, ses croyances"

[Clancey, 1991a, p. 392]

Le second point est tout à fait lié au premier :

b) Lorsqu'un humain *pense*, il se passe *quelque chose d'autre* que la simple exécution de plans préétablis ([Suchman, 1987]).

"L'action humain n'est pas, comme le suggère le point de vue 'modélisation descriptive', une combinaison astucieuse de plans qui se suivent ou d'action située ; au contraire, suivre attentivement un plan requiert de reconcevoir ce qu'il signifie au cours de l'activité elle-même, c'est cela l'action située"

[Clancey, 1997a]

"Pour récapituler, ma formulation de l'hypothèse de la cognition située est que tous les processus de comportement, y compris la parole, la résolution de problème, et les aptitudes physiques, sont générés sur le champ, et non par l'application mécanique de scripts ou règles stockés préalablement dans le cerveau"

[Clancey, 1991b, p. 110]

Ce "quelque chose qui se passe en situation", lorsqu'on comprend, lorsqu'on parle, lorsqu'on agit, Clancey le nomme diversement :

(re)conceptualisation / création / conception / apprentissage / (re)composition, ...

de : structures / coordinations / motifs émergents / couples perception-action,...

Les quatre extraits suivants témoignent de la variété des descriptions de ce phénomène, qui n'est pas loin d'être le cœur de l'activité cognitive :

"[Une description au niveau Connaissances] se rapporte aux motifs qui émergent dans les interactions que l'agent met en oeuvre avec un environnement (social)"

[Clancey, 1991a, p. 361]

"L'action située implique de manière inhérente l'apprentissage de nouvelles coordinations au cours de chaque interaction"

[Clancey, 1993a, p. 104]

"[Situé signifie] développer de manière adaptée de nouvelles structures au cours de ses interactions avec son environnement"

[Clancey, 1993b, p. 37]

"... les 'retours-dans-l'instant' (feedback-in-the-moment) tactiles et visuels sont essentiels pour la recoordination inventive, qui fait partie du raisonnement de tous les jours"

[Clancey, 1997b, p. 196]

En fin de compte, ce "quelque chose", ce "type d'intelligence", l'IA n'arrive pas à le capturer :

"Le couplage direct dans le cerveau entre les processus perceptuels, conceptuels et moteurs nécessite une sorte 'd'auto-organisation avec mémoire' que nous n'avons pas encore reproduite dans des programmes informatiques, ni dans aucune machine"

[Clancey, 1997b, p. 2]

On a même le droit de douter qu'elle y arrive un jour. Et c'est un peu la question qui est posée à l'IA : quelles sont les conséquences à tirer de tout cela ? Est-ce regrettable de ne pas parvenir à intégrer cette "intelligence située" dans les programmes d'ordinateurs ? Nous allons nous prononcer sur ces questions.

W. Clancey, quant à lui, tire de la cognition située deux recommandations pour les concepteurs de systèmes experts :

"1) Ne vous contentez pas de transmettre la technologie ; travaillez sur les sites en collaboration avec les chercheurs en sciences humaines et les utilisateurs pour des conceptions fortement incrémentales.
2) Facilitez les conversations, ne les automatisez pas seulement"

[Clancey, 1993b, p. 44]

Plus généralement pour Clancey, la cognition située donne à l'informatique des objectifs nouveaux qui sortent du cadre traditionnel de l'acquisition des connaissances :

"... nous ne voyons pas le rôle de l'ingénierie des connaissances comme la 'capture de connaissances' dans un programme livré par des techniciens à des utilisateurs. Au contraire, nous cherchons à développer des outils qui aident les personnes d'une communauté dans leur pratique quotidienne de création de nouvelles manières de comprendre, de nouvelles capacités, de nouvelles formes de connaissances"

[Clancey, 1993b, p. 49]

"En résumé, la cognition située [.../...] requiert une vue dynamique, transactionnelle de l'expertise et des outils. Le point essentiel est de faciliter la construction de connaissances en facilitant les conversations et en utilisant les outils de modélisation de telle sorte que les gens puissent exprimer leurs points de vue, mener des implications logiques et comparer des alternatives"

[Clancey, 1997b, p. 367]

[Sandberg et Wielenga, 1991] tirent de la cognition située des leçons similaires à ce que nous avons développé dans la section 1.1.4 sur la nécessaire prudence requise lors des processus d'abstraction et de généralisation :

"Une des conséquences du point de vue situé est de se retenir de décontextualiser, d'abstraire à partir des particularités d'une situation de problème."

[Sandberg et Wielenga, 1991, p. 345]

[Menzies, 1996] dégage différentes réponses et champs de recherche possibles face à la cognition située, du point de vue du domaine de l'acquisition des connaissances. Il commence par dégager deux conceptions de la cognition située. En premier lieu il présente une conception dite "faible" de la cognition située, derrière laquelle il se range :

"La cognition humain ne peut pas être modélisée précisément par des assertions indépendantes du contexte parce que les inférences d'un modèle symbolique interagissant avec son environnement sont fortement contraintes / contrôlées / changées par les données provenant de cet environnement. Autrement dit, utiliser des connaissances dans un contexte particulier modifiera ces connaissances de manière significative"

[Menzies, 1996]

La cognition située dite "forte" rassemble par exemple, les roboticiens refusant la notion même de représentation ([Brooks, 1991]). Tim Menzies résume ce point de vue, qui par ailleurs n'est pas compatible avec les objectifs de l'acquisition des connaissances, de la manière suivante :

"Puisque l'influence de l'environnement est si forte, nous devons utiliser des systèmes purement réactifs qui interagissent directement avec l'environnement sans réfléchir sur des descriptions symboliques"

[Menzies, 1996]

Parmi les premières réponses possibles en acquisition des connaissances à la cognition située dite "faible", T. Menzies distingue :

- les Règles Dé-Roulées (pour Ripple-Down Rules), que nous présenterons dans la section 4.1,
- les grilles-répertoires,
- les outils de vérification et de validation,
- les systèmes critiques,
- l'apprentissage,
- les systèmes d'aide à la décision,
- et bien sûr, le choix d'ignorer complètement ce que dit la cognition située...

Toutes ces réponses (sauf la dernière) ont pour point commun un souci de chercher à corriger le modèle de l'expertise construit et représenté au sein du SBC :

"L'ensemble de ces travaux met l'accent sur le fait que la capacité à évaluer et modifier un modèle est plus importante que la capacité à le spécifier initialement"

[Menzies, 1996]

Mais aucune de ces réponses n'est à nos yeux pleinement satisfaisante. Ce sont toutes en effet des réutilisations de techniques ou d'outils existants et dont les objectifs ne cadrent jamais complètement avec la problématique fixée par T. Menzies : comment concilier cognition située dite "faible" et acquisition des connaissances ? Par exemple, les grilles-répertoires sont des outils d'élicitation des connaissances, mais pas de révision des connaissances...

Or, la démarche que nous nous sommes fixée depuis la section 1.1 est aussi une proposition de réponse à cette problématique. Le lecteur a déjà pu remarquer que l'idée de système évolutif s'appuyait sur les mêmes bases que les idées de W. Clancey présentées au point a) : une BC ne peut capturer "l'intégralité d'une expertise". Dans la section suivante, nous allons développer une réponse au problème soulevé dans le point b) (comment faire face à "l'intelligence qui émerge en situation"), cohérente avec notre démarche : nous l'appellerons "révision située".

1.2.3.2 Notre interprétation : la révision située

Nous résumerons notre interprétation de la cognition située, pour l'IA et l'acquisition des connaissances en particulier, à deux idées :

1) Un Système à Base de Connaissances doit toujours être "ouvert". Nous retrouvons exactement le concept de système évolutif que nous avons déjà développé et qui portait l'objectif de révisabilité au cœur de notre démarche. Le couple environnement / système cognitif est lié d'une manière étroite et dynamique, et le SBC, qui est une modélisation prescriptive du système cognitif pris isolément, aura toujours un "temps de retard" sur cette dynamique. On ne peut y représenter que les "coordinations système-environnement", les "retours", décelés et explicités par le concepteur. Il faut donc se préparer à constamment être capable de corriger la BC.

"En résumé, la cognition située est l'étude de la manière que possède la connaissance humaine de se développer comme un moyen de coordination de l'activité à l'intérieur de l'activité elle-même. Cela signifie que le retour — qui a lieu en interne et avec l'environnement dans le temps— est d'une importance primordiale"

[Clancey, 1997b, p. 4]

2) L'essentiel de la question ("l'intelligence") réside dans la capacité à gérer de nouvelles situations en temps réel. Ce deuxième point, lié au point b) de la section précédente ne rentre a priori pas dans la démarche que nous nous étions fixée... Effectivement, nous voulons un système qui se comporte exactement comme prescrit dans les situations déjà rencontrées ou déjà décrites. Nous n'avons pas formulé d'exigence de comportement pour les situations jamais encore rencontrées ou décrites. Or l'idée développée par W. Clancey et la cognition située, c'est qu'il existe une composante inventive, une capacité d'improvisation en situation qui est une part fondamentale du comportement humain.

"La cognition située en elle-même n'est pas une prescription pour apprendre (apprentissage situé), mais plutôt une affirmation / proposition que l'apprentissage a lieu avec chaque comportement humain. Dans la coordination physique — et la véritable nature de la mémoire — les actions sont toujours à un certain niveau improvisées."

[Clancey, 1997b, p. 344]

On peut détecter une antinomie possible entre cette capacité d'improviser, d'inventer et notre exigence de précision. En effet, la flexibilité / plasticité, qu'on sait caractéristique du cerveau, et qui semble entrer en jeu dans cette intelligence

située peut être opposée au besoin d'avoir un système donnant uniquement des réponses validées. En poursuivant cette opposition, on peut esquisser deux types de systèmes, en fonction de l'exigence de prescriptivité (est-il important ou pas de donner des réponses strictement validées ?) et la disponibilité d'un utilisateur / tuteur.

Les deux temps du fonctionnement du premier type de système seraient les suivants :

1) simplement **stocker** les expériences exactement telles qu'elles sont rencontrées,

2) lors de la résolution de problème, être capable de réfléchir "tout seul" à partir du stock d'expériences : **improviser**.

Ce premier type de système, qu'on peut qualifier de "**système autonome**", n'a pas besoin d'un tuteur pour fonctionner, contrairement au second type de système (qu'on pourrait alors appeler "**système obéissant**"). Les deux temps du fonctionnement du second type de système sont les suivants :

1) **intégrer** les expériences rencontrées avec l'aide d'un tuteur, c'est-à-dire généraliser les connaissances et en préciser toutes les conséquences pratiques immédiatement.

2) lors de la résolution de problème, se contenter d'**exécuter** ce qu'on a appris antérieurement.

Les systèmes de raisonnement à partir de cas peuvent être classés grossièrement dans la catégorie des systèmes autonomes :

"... pour des raisons pratiques, un expert du domaine voudra pouvoir entrer beaucoup de cas dans un RPC sans avoir à spécifier les raisonnements sous-jacents"

[Riesbeck et Schank, 1989, p. 44]

Leur grande force est d'être capable d'être suffisamment souples pour proposer des réponses à des situations inédites.

"Un système de RPC sera restreint à des variations sur des situations connues et il produira des réponses approximatives, mais il sera rapide et ses réponses seront enracinées dans l'expérience"

[Riesbeck et Schank, 1989, p. 26]

A l'opposé, la démarche exposée dans la section 1.1 nous place dans la catégorie de ceux qui veulent construire des systèmes obéissants... Nous voulons un système capable de faire exactement ce qu'on lui dit, d'obéir : d'apprendre précisément, puis de fonctionner ensuite sans surprises pour son concepteur,... Avec

cette vue, le système garde avant tout un statut d'outil, avant de prendre un statut d'agent. Nous défendons l'idée que **c'est là une forme d'intelligence importante que d'être capable d'apprendre correctement**. Le terme "système obéissant" ne doit pas être pris de manière péjorative ou réductrice puisque le terme recouvre une "intelligence d'anticipation", une capacité à intégrer et de généraliser de manière adéquate, qui est primordiale pour le fonctionnement du système. En effet, dans notre approche, le comportement futur est entièrement déterminé par les généralisations effectuées durant l'étape d'intégration / apprentissage. L'apprentissage "anticipé", c'est-à-dire provoqué par la détection d'une insuffisance, mais dirigé vers l'avenir, est une réponse intéressante au problème de l'intelligence située.

Bien entendu, il est sans doute possible de concilier les deux approches en conservant notre objectif de prescriptivité et de validation des apprentissages du système : on peut proposer d'enrichir un système obéissant par des capacités d'improvisation à condition que le système soit capable de détecter les situations inédites. On distinguerait alors quatre étapes :

1) **stocker** les expériences,

2) **apprendre** à partir de ces expériences (avec l'aide ou non d'un tuteur),

3) **exécuter** si la situation est connue ou **improviser** si la situation est inédite,

4) **évaluer** l'éventuelle improvisation (avec l'aide ou non d'un tuteur), et boucler en fonction de cette évaluation sur l'étape 2 (intégrer l'improvisation) ou l'étape 3 (proposer une nouvelle improvisation)[11].

Mais dans le cadre de ce travail, nous en restons à une démarche de construction de "systèmes obéissants". Notre priorité reste avant tout de chercher à construire un système capable de "recevoir" et de digérer des connaissances avant d'être capable d'imaginer[12]... Notre slogan pourrait être : "l'imagination ne sert à rien sans capacité d'apprentissage".

Nous nous situons finalement dans le cadre de ce que nous proposons d'appeler la **révision située** et dont nous résumons la problématique de la manière suivante :

---

11 Dans tous les cas, il est important que le système sache si la solution qu'il propose est une *réactivation*, ou une *construction* (improvisation). D'abord parce que c'est une donnée pour connaître le risque pris en proposant la solution, mais surtout parce que sinon, il est incapable d'apprendre... Contrairement à l'acceptation d'une réactivation, l'acceptation d'une construction nécessite une révision (intégration de la nouvelle solution). Le rejet d'une construction nécessite une nouvelle construction; le rejet d'une réactivation, une révision.

12 Rappelons toutefois qu'un système obéissant est aussi capable d'imaginer et de proposer des généralisations après la première étape d'intégration. Le critère déterminant qui distingue système autonome et système obéissant, c'est le moment de cette nouvelle construction qui fera de celle-ci une improvisation (système autonome) ou une proposition d'anticipation (système obéissant) .

Pour un cas (problème) donné, l'utilisateur donne (ou corrige) une solution que le système "apprend" (intègre).

Problématique de la révision située

La révision située est le terme générique pour résumer notre démarche, qui a été explicitée dans la section 1, et que nous enrichissons à la lumière des réflexions autour de la cognition située et des systèmes apprentis.

Au cours de notre démarche, nous avons placé la révisabilité au coeur de nos objectifs : nous sommes allé jusqu'à proposer de construire une représentation des connaissances **orientée révision,** c'est-à-dire destinée à faciliter la révision.

Notre interprétation de la cognition située nous confirme l'importance de la capacité à réviser la BC à tout moment, et notamment amène l'idée qu'il y aura toujours des défaillances, ruptures, erreurs, insuffisances du système qui seront mises au jour "en situation", au cours du fonctionnement du système.

Un outil de révision coopérative doit aider le concepteur de la BC à corriger la défaillance, tout en faisant très attention à faire une correction locale, ou du moins à faire une correction dont les conséquences, si elles dépassent le cas "problématique" initial, sont parfaitement comprises et validées par l'utilisateur.

On retrouve ce principe de "révision prudente" dans le système APT :

"Ainsi, le système doit aider l'expert à identifier correctement la cause du problème dans la base de connaissances, et ensuite proposer des modifications en utilisant le contexte du problème, et en montrant à l'expert toutes les conséquences des modifications proposées sur le comportement du système"

[Nédellec et Causse, 1992, p. 173]

Nous adhérons complètement à la stratégie générale adoptée par le système APT pour communiquer avec l'expert :

"Demandez uniquement à l'expert d'évaluer et comparer des cas, et non des connaissances générales"

[Nédellec et Causse, 1992, p. 173]

Cette stratégie correspond à "l'heuristique des situations spécifiques", que définit Patrick Winston pour l'ingénierie des connaissances ([Winston, 1992]). Cette heuristique peut être exposée de la manière suivante : ne pas simplement poser la question générale à l'expert "comment procédez-vous dans votre travail ?", mais le regarder travailler et lui poser des questions abondantes et précises lorsque des situations spécifiques surviennent. L'intérêt des systèmes apprentis est de

permettre d'automatiser ce procédé. Cette stratégie rejoint quelque part le point de vue “cognition située” où l'accent est mis sur l'étude des activités et sur la pratique ([Lave, 1988]).

En résumé, le projet de “révision située” propose de construire des systèmes apprentis avec les caractéristiques suivantes :

- la représentation des connaissances utilisée fait en sorte que la BC soit compréhensible et que les révisions effectuées puissent être faciles et locales.

- un moteur ou outil de simulation de la BC basé sur cette représentation, doit permettre à l’utilisateur de comprendre les cas de résolution et de les stocker éventuellement en vue d'une révision,

- un outil de révision coopérative est capable, avec l’aide du concepteur de la BC, de corriger l’erreur, de valider cette correction au moyen d'exemples concrets soumis à l'utilisateur, puis de généraliser prudemment la correction.

1.3 Le cheminement vers le modèle

L'inventaire que nous venons d'effectuer ne nous a finalement fourni aucun concept ou outil qui remplisse complètement nos objectifs. La section 1.2 nous a permis de compléter et préciser nos objectifs, de donner des pistes de solutions, mais elle ne nous a pas donné de modèle “clé en main” à réutiliser. Par conséquent nous voilà amenés à forger notre propre solution. Nous exposons dans cette section le cheminement qui va nous conduire vers notre modèle de représentation des savoir-faire.

Nous allons proposer un modèle simple, opératoire, pragmatique, dont la construction est guidée par :

- les objectifs que nous nous sommes fixés,

- les concepts et outils “ambiants” en IA, en particulier ceux que nous avons présentés dans la section 1.2, qui vont énormément influencer la construction de notre modèle,

- une réflexion, un point de vue personnel, une intuition introspective sur la manière dont on raisonne.

Ce dernier élément est bel et bien présent et motive la proposition d'un modèle original de représentation des connaissances. Il ne faut donc pas l'ignorer et le passer sous silence sous prétexte de “non-scientificité”. L'importance de la dimension intuitive et introspective dans la construction de modèles d'IA est une des spécificités de l'IA, qui participe à son charme. Cela vient des buts particuliers que

sont la représentation et la manipulation des connaissances. Si l'on veut rendre compte du raisonnement de l'expert, on doit bien s'intéresser à la nature de son raisonnement. C'est là bien évidemment un parti-pris qui résulte de nos objectifs de compréhensibilité "permanente", c'est-à-dire à tout moment durant le fonctionnement du système. Nous aurions évidemment tout aussi bien pu décider de nous éloigner de ce type de démarche qui peut être qualifiée de "cognitiviste" parce qu'elle fait l'hypothèse d'une certaine correspondance entre représentations mentales et représentations informatiques.

Précisons tout de suite que le modèle qui va être proposé n'aura pas prétention de théorie cognitive ou psychologique. Pour au moins deux raisons :

1) Il s'agit très simplement comme nous le verrons d'un enrichissement du modèle Stimulus-Réponse, accompagné de l'exposé d'un point de vue particulier sur la notion de situation. La simplicité et l'évidence du modèle et de ses composantes ne motive pas de validation de la psychologie cognitive.

2) Et surtout, ce modèle n'a pas été construit dans le but de modéliser l'ensemble des activités cognitives et d'être valide du point de vue de la psychologie cognitive. Il faut bien avoir en mémoire que, conformément à nos objectifs, nous cherchons à bâtir un modèle **prescriptif** du raisonnement, ce qui réduit considérablement l'ambition de notre modèle.

1.3.1 Le rôle de l'environnement

Lorsque l'on commence à réfléchir sur la cognition en général, il est important de bien prendre en considération le rôle fondamental que joue l'environnement dans tout processus cognitif. Par environnement, nous entendons tout ce qui est hors-système (le monde extérieur).

"C'est uniquement parce que le cerveau subit des interactions avec l'environnement que nous pouvons qualifier de cognitif le comportement qui en résulte."

[Varela, Thompson et Rosch, 1993, p. 36]

Le rôle de l'environnement a été longtemps sous-évalué en IA, où l'on construisait des systèmes fermés définitifs :

- *fermés* au sens où peu d'interactions avec l'environnement ont lieu au cours de la résolution du problème : les données en provenance de l'environnement sont souvent considérées comme déjà connues au commencement de la résolution.

- *définitifs* au sens donné dans la section 1.1.2 : le système n'a pas besoin de faire évoluer ses connaissances, parce qu'on considère que sa BC est correcte et complète.

Les travaux en robotique et autour de la notion d'agent, ainsi que les idées issues de la cognition située (cf. section 1.2.3), ont remis au goût du jour la nécessité de concevoir des systèmes ouverts sur l'environnement, capables de s'adapter et d'apprendre au fur et à mesure des expériences effectuées.

1.3.2 Du schéma Stimulus-Réponse au schéma Situation-Action

La cognition commence quand le système possède la capacité d'identifier des composantes de l'environnement, ce qui lui permet éventuellement de réagir à ces composantes. Cette capacité se traduit primitivement et simplement sous forme de réflexes, innés ou appris.

C'est le schéma Stimulus-Réponse qu'on trouve donc au départ (et à la base) de la cognition. Ce qui permet le fonctionnement du schéma Stimulus-Réponse, c'est :

a) une "**sensibilité à l'environnement**". L'identification de composantes de l'environnement peut être élaborée et volontaire (on parle alors de perception active), mais elle peut tout aussi bien être automatique dans le cas du schéma Stimulus-Réponse (perception passive).

b) une possibilité d'**agir sur l'environnement**. La cognition est vaine sans possibilité d'agir. On peut considérer que l'action finalement effectuée est la finalité du processus cognitif.

Il est possible de généraliser ce schéma Stimulus-Réponse très simple si l'on introduit la notion de **projet** (ou de but). L'introduction de la notion de but fait passer l'agent du statut d'agent *tropique*, c'est-à-dire dont le comportement est entièrement déterminé par l'état de l'environnement, au statut d'agent *hystérétique*, qui doit, afin d'être capable de mémoriser un but, être doté d'une mémoire interne ([Genesereth et Nilsson, 1987]).

On étend alors le schéma primitif Stimulus-Réponse à un schéma plus général et plus complexe que l'on appellera schéma Situation-Action. Selon ce schéma, toute connaissance doit pouvoir s'exprimer de la manière suivante :

"QUAND je suis dans la situation S,
IL FAUT (ou IL EST RAISONNABLE d') effectuer l'action A".

1.3.3 La notion de situation

La notion de situation telle qu'elle apparaît dans ce schéma Situation-Action représente à la fois des connaissances sur le monde *et* un projet en cours (une tâche à accomplir). La figure 1.2 illustre ces deux dimensions qui sont constitutives de la notion de situation.

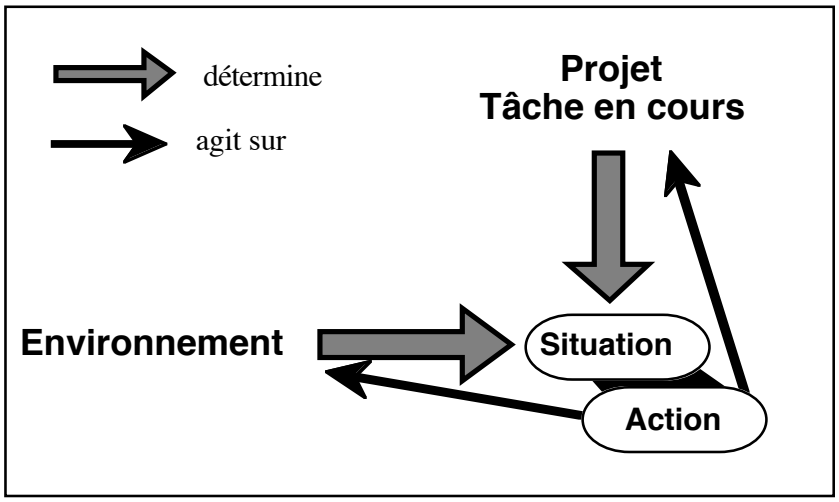


Figure 1.2 : Le schéma projet- environnement

Pourquoi choisir le terme *situation* pour qualifier cet objet qui est à la fois une interprétation de l'environnement et une étape au cours de l'accomplissement d'une tâche ? Il est vrai que nous nous opposons alors à la vision courante de la situation comme notion objective. Or nous défendons l'idée que la situation est une notion éminemment subjective, puisqu'elle est rattachée à un sujet, et dépend d'une part des informations sur le monde dont il dispose, d'autre part des buts, des intentions du sujet.

Appelons les philosophes à la rescousse...

"La situation factuelle implique l'acte interprétatif d'un sujet : l'acte de situer. Situer, c'est mettre une forme intelligible (un ensemble de rapports) dans l'expérience temporelle multiple et muette, c'est mettre en perspective une multiplicité désordonnée et confuse. Toute situation est donc l'éclairage des faits à partir d'un sens constitué par l'intérêt de celui qui en parle."

[Robinet, 1990]

"Le concept de situation est caractérisé par le fait qu'on ne se trouve pas en face d'elle, qu'on ne peut avoir d'elle un savoir objectif. On est toujours placé dans une situation, on s'y trouve déjà impliqué et l'éclaircissement de cette situation constitue la tâche qu'on arrive jamais à achever"

[Gadamer, 1976, p. 142]

Donc pour nous (et pour notre modèle de représentations prescriptives), une situation n'a de sens que rapportée à un projet. Il convient alors de distinguer la connaissance de l'environnement (connaissance "objective" de l'état du monde), de la notion de situation définie ici comme **l'interprétation de l'environnement au cours de l'accomplissement d'une tâche** : c'est-à-dire une connaissance de l'état du monde, associée à un projet en cours.

La notion de projet[13] qui apparaît avec notre notion de situation amène avec elle les notions de temps et de mémorisation que le schéma Stimulus-Réponse n'intégrait pas. La réalisation d'un projet nécessite une planification et un suivi : l'accomplissement d'une tâche est découpée en étapes. A chaque étape, le mécanisme Situation-Action entre en scène. La différence avec le schéma Stimulus-Réponse est que l'action exécutée n'est pas une réponse "finale", mais renvoie des informations qui sont utiles au processus global. La réponse "finale" ne sera donnée qu'à la fin de ce processus. En fait ces informations renvoyées modifient la situation elle-même. Le modèle de raisonnement que l'on propose peut être résumé comme formulé dans la figure 1.3.

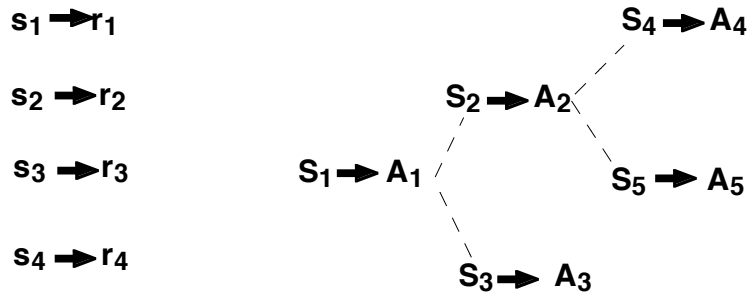
"Dans une situation donnée, il convient d'effectuer certaines actions sur l'environnement. Celles-ci peuvent éventuellement renvoyer des informations qui participent à la redéfinition ou précision de la situation."

Figure 1.3 : Le modèle de raisonnement du schéma Situation-Action

On arrive ainsi à l'idée que l'accomplissement d'une tâche est un *enchaînement successif de situations* à chaque fois légèrement différentes. On peut aussi considérer que la situation est un objet au départ très général, qui s'affine au fur et à mesure de l'avancement de la tâche et de la découverte d'informations sur l'environnement. Les deux points de vue sont possibles, selon qu'on considère que le changement opéré à chaque étape (le nouvel "éclairage" apporté par les informations) est important et définit un nouvel objet (la situation change) ou que l'on considère que ce changement ne fait que compléter et préciser la vision précédente de la situation (la situation s'instancie). L'essentiel est d'admettre le processus général d'affinements successifs qui mène d'une situation initiale exprimant un projet, parfois sans connaissance sur l'environnement, à une situation dite *finale*, où l'environnement est alors "connu", au sens où l'on est allé chercher toutes les informations dont on avait besoin pour réaliser notre projet. L'action

[13] Nous préférons employer le terme "projet" plutôt que le terme "but", dont la définition usuelle en IA ne concorde pas exactement avec notre approche. Nous reviendrons sur cet aspect dans la section 1.3.4.

associée à la situation finale est l'action “adéquate” : celle qui correspond le mieux au couple “projet initial” / “état de l'environnement”. C'est la conclusion du raisonnement, la réponse finale donnée à l'environnement.



Le schéma Stimulus--Réflexe: des ré-actions instantanées indépendantes les unes des autres

Le schéma Situation-Action : le fil d’un raisonnement

Figure 1.4 : Schéma comparatif des deux modèles

Le modèle Situation-Action propose finalement la définition opérationnelle suivante de la connaissance :

Une connaissance est la spécification d’une association

entre une situation et une action.

Cette définition donne au réflexe le statut de “connaissance la plus primitive”. C'est l'association directe d'une situation dans sa forme la plus simple (un stimulus ou un ensemble de stimuli) et d'une action.

Le fait d'associer une action à effectuer pour chaque situation est la marque du caractère prescriptif de notre modèle (et non un symptôme de dérive comportementaliste). Il faut rappeler que l'on veut *représenter des connaissances*, donc :

1) exprimer concrètement *ce qu'il faut faire* dans chaque situation référencée, et

2) ne référencer que les situations significatives, c'est-à-dire les situations qui rentrent dans le champ de la tâche à exécuter.

Cette association Situation-Action trouve des échos en philosophie.

"Le constat interprétatif trouve l'accomplissement de son sens dans l'action qu'il conditionne"

[Robinet, 1990]

"La situation, lorsqu'elle est connue, appelle un comportement"

[Jaspers, 1932, p. 23]

Karl Jaspers poursuit :

"L'appréhension de la situation est d'une telle nature qu'elle a déjà changé, aussitôt qu'elle rend possible l'appel à l'action et à un comportement"

[Jaspers, op. cit.]

Cette dernière phrase évoque la citation de Hans-Georg Gadamer ci-dessus qui faisait référence au processus toujours inachevé "d'éclaircissement de la situation". H.-G. Gadamer poursuit en indiquant que :

"cet inachèvement ne tient pas à un manque de réflexion, mais résulte de l'essence de l'être historique que nous sommes".

[Gadamer, op. cit.]

Notre modèle discrétise ce processus d'éclaircissement en proposant de représenter explicitement ses différentes étapes. Mais encore une fois, nous nous limitons à *prescrire* des connaissances mises en oeuvre pour la réalisation d'un projet. Par cette réduction, notre modèle n'est pas concerné par le problème de la compréhension du monde et de la genèse des projets.

1.3.4 Le nodule de situation

Comment représenter concrètement les situations et ce schéma Situation-Action dans un SBC ?

Trois caractéristiques définissent une situation :

1) Une étape au cours de l'accomplissement d'une tâche. Une situation prend son sens relativement à d'autres situations. Elle est définie par une **position** dans un réseau de situations, tel que celui représenté figure 1.4. Il est difficile de donner à la situation une définition en extension ou dans l'absolu. Il est beaucoup plus aisé de la définir relativement à d'autres situations, c'est-à-dire relativement à celles qui la précèdent et qui la suivent dans le réseau.

2) L'**action** associée, bien sûr. Nous parlerons ici d'action au singulier, mais il est bien sûr possible d'associer plusieurs actions successives à une situation.

3) Enfin, des **liens** avec les situations suivantes, celles qui doivent être "inférées" à partir des informations renvoyées par l'action. Nous choisissons d'associer cette "connaissance pour choisir la situation suivante" à chaque situation.

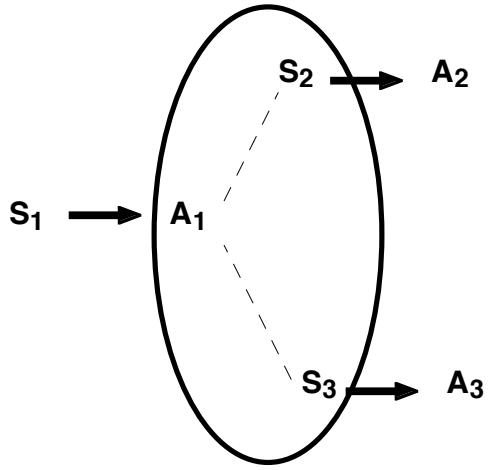


Figure 1.5 : Le nodule de situation dans le schéma Situation-Action.

L'idée est donc de **réifier la situation** sans chercher à lui donner une définition formelle in extenso, mais de la définir, en s'appuyant sur le schéma Situation-Action, comparativement aux autres situations. L'objet qu'on va appeler *nodule*[14]*de situation* possède un identifiant unique et est accessible par cet identifiant dans une base de nodules. Il contient **une action** et **des règles de choix** de la situation suivante. La figure 1.5 schématise ce que modélise un nodule de situation dans le schéma Situation-Action. Le nodule de situation représente l'action et le lien vers les situations suivantes. Il prend son sens de par sa position dans le réseau de nodules.

Nous choisissons de représenter le "réseau de nodules" qui apparaît dans la figure 1.4, sous la forme d'un ensemble de graphes de nodules. Un graphe de nodules est identifié par son nodule initial, qui représente un but, un projet, ou mieux : une *attente*. En effet, la notion de but en IA est en général une expression formelle d'un état désiré, ce qui dans notre modèle n'est pas le cas du nodule initial. Le nodule initial représente plutôt une direction donnée, l'objectif d'exécuter une certaine tâche (par exemple "donner les coordonnées du bureau compétent" ou "permettre à l'utilisateur de saisir un message"). Certes le nodule initial, et à travers lui l'ensemble du graphe, prédétermine un ensemble de solutions possibles : l'ensemble des nodules terminaux du graphe. La situation-attente initiale se précise

[14] Du latin *nodulus* pour petit nœud.

au fur et à mesure de l'avancement de la tâche, jusqu'à devenir une situation-solution. C'est à J. Lave que nous empruntons le terme *attente* ("expectation") :

> *"La formation, la tenue d'un rôle et l'activité se rejoignent dans la construction et la réalisation de ce qu'on pourrait appeler des attentes plutôt que des buts. Celles-ci devraient être pensées comme des formes de résolution potentielle incarnées dans l'activité vécue durant la formation. Les attentes [.../...] rendent possible l'activité tandis qu'elles changent au cours de l'activité, à la fois en avant et en arrière dans le temps"*
>
> [Lave, 1988, p. 184]

Nous assimilons donc nodule initial et projet (ou attente), mais aussi graphe et tâche. Une tâche est définie par un projet (nodule initial), mais aussi par les activités qui contribuent à la réalisation de ce projet : ces activités sont représentées par la totalité des nodules du graphe. La distinction classique en acquisition des connaissances entre tâche et méthode permet de distinguer les buts ou sous-buts (tâches et sous-tâches) des moyens employés pour atteindre ces buts (méthodes). Nous ne faisons pas cette distinction car nous ne cherchons pas à représenter explicitement les buts à atteindre indépendamment des activités qui permettent d'atteindre ces buts. Autrement dit, lorsque nous employons le terme tâche dans notre approche, il s'agit de tâche "opérationnelle", au sens où le savoir-faire pour accomplir cette tâche (la méthode) est connu et attaché à la tâche. De façon similaire, nous n'employons pas le terme *problème* car nous nous attachons uniquement à représenter la *manière* de résoudre les problèmes, et nous préférons par conséquent parler de *tâches*.

La contrainte de représentation séquentielle du raisonnement (section 1.1.3) nous pousse à exiger que les nodules ne puissent être activés que successivement à partir du nodule initial. A un instant donné, un seul nodule est activé : il est dit nodule courant et représente la situation courante. Il exprime à la fois l'activité passée (l'historique des nodules qui ont été activés) et le projet courant qui a pu se spécialiser depuis le nodule initial.

L'algorithme général du moteur exploitant les nodules de situation est le suivant :

Lorsqu'un nodule est activé, toutes ses actions sont exécutées. Celles-ci peuvent renvoyer des faits plus ou moins structurés, qui sont stockés dans une Mémoire de Travail (MT). A partir de cette MT, les règles de choix du nodule courant permettent de choisir un nodule successeur. Si un nodule successeur est

trouvé, il est activé à son tour. Sinon, le nodule courant est dit final et l'algorithme se termine.

Figure 1.6 : L'algorithme de fonctionnement du moteur

On distinguera donc la notion de *nodule terminal* d'un graphe (nodules n'ayant pas de successeurs dans le graphe), de la notion de *nodule final*, relative à une exécution pour un environnement donné[15].

1.3.5 Faits et Mémoire de Travail (MT)

1.3.5.1 Hypothèse du monde fermé et principe de perception active

La Mémoire de Travail sert à recueillir provisoirement les *faits* connus de l'environnement, qui ont été renvoyés par l'exécution des actions. Les prémisses des règles de choix des nodules portent sur ces faits.

Tout fait qui n'est pas présent dans la Mémoire de Travail au moment où celle-ci est consultée (i.e. lors du choix du nodule suivant) est considéré comme faux.

Hypothèse du monde fermé

L'hypothèse du monde fermé donnée ci-dessus requiert une petite explication. Le schéma Situation-Action dissocie la "recherche d'information dans l'environnement" (ou *consultation*), effectuée par les actions, de "l'utilisation de cette information", spécifiée par les règles de choix. Mais ces deux aspects de la résolution de problème, *consultation* et utilisation, n'en sont pas moins très étroitement liés, au sein d'un même nodule. L'intérêt de cette dissociation est d'insister sur l'importance de l'aspect *consultation*. Le principe de "perception active" formulé ci-dessous résume cette obligation de toujours spécifier explicitement *l'acte* de recherche d'information :

[15] Précisons qu'un nodule final n'est pas toujours terminal : en conséquence de notre approche de construction incrémentale de graphes de nodules de situation, il est à chaque instant possible de tomber sur un nouveau cas, pour lequel le graphe ne donne pas de nodule terminal. Si le concepteur de la BC s'est efforcé de toujours associer un nodule terminal aux différents cas prévus, c'est alors le symptôme d'un cas imprévu, pour lequel l'action finale n'a jamais été validée par le concepteur. C'est une propriété intéressante du système que d'être ainsi capable de détecter son incompétence pour le cas courant ([Compton, Preston, Edwards et Kang, 1996]). Toutefois, parvenir à un nodule terminal n'est inversement pas une assurance que le cas courant ait été validé.

Une information doit être “pêchée” avant d'être utilisée.

Principe de perception active

Le principe de perception active illustre le fait que, dans notre modèle, les informations ne sont jamais données à l'avance mais que la recherche d'information dans l'environnement fait partie intégrante du raisonnement.

Notons que la dissociation entre consultation de l'environnement et utilisation de l'information distingue fondamentalement notre modèle du modèle dit de “calcul de situation” ([MacCarthy et Hayes, 1969]). Dans le calcul de situation, comme une situation est la description formelle d'un état de l'environnement, l'exécution d'une action sur l'environnement provoque de façon immédiate un changement de situation... Il n'y a dans ces conditions pas de distinction possible entre action et choix de la situation suivante : cette dissociation dans notre modèle est la conséquence de notre conception de la notion de situation comme une notion subjective, distincte de l'état de l'environnement[16].

1.3.5.2 Monotonie du raisonnement

La question de la persistance des faits de la Mémoire de Travail se pose. Une alternative simple se dessine :

- ou bien la MT est locale au nodule activé, et aucun fait n’est conservé d’un nodule à l’autre,

- ou bien les faits stockés dans la MT sont conservés jusqu’à la terminaison du moteur.

La deuxième solution est retenue faisant l'hypothèse que nous concevons un modèle pour des environnements invariants. Entre le démarrage du moteur et sa terminaison (l’exécution de l’action finale), nous considérons que l’environnement n’a pas changé, donc que les informations obtenues sur l’environnement restent valables du début à la fin. Nous pourrons dire par conséquent que le raisonnement représenté est monotone, puisque dans ces conditions aucun nouveau fait ne pourra jamais contredire un fait déjà connu et déjà utilisé.

1.3.5.3 Définition des actions

Dans un premier temps, nous considérerons que les faits stockés dans la MT sont tous propositionnels et nous ne distinguerons pas d'objets dans le monde

16 De même, notre travail se distingue de la théorie des situations ([Barwise et Perry, 1986], [Devlin, 1991]), pour laquelle : “Une situation est une portion limitée du monde (sur un lieu ou un intervalle de temps), qui peut être choisie par un agent cognitif” [Akman et Surav, 1996].

réel, possédant des propriétés et des relations. Par exemple l'action *regarderSiPluie* peut renvoyer deux faits, *ilPleut* ou *ilNePleutPas*. La liste de l'ensemble des faits que peut alternativement renvoyer chaque action doit être déclarée par le concepteur de la BC. Elle constitue la *définition* de l'action. Il est effectivement nécessaire de connaître les faits susceptibles d'être renvoyés par les actions pour être en mesure de construire les règles de choix s'appuyant sur ces faits.

1.3.5.4 Règles de choix des nodules

Les règles de choix sont indépendantes entre elles. Elles sont de la forme : "SI *prémisse* ALORSACTIVER *nodule*". Elles permettent de sélectionner ou non un nodule successeur. Si aucune règle n'est activable, le nodule courant devient nodule final et l'algorithme se termine. Si plusieurs règles sont simultanément activables, un système de priorités permet de choisir l'une des règles, donc l'un des nodules candidats, qui sera immédiatement activé à son tour.

1.3.6 Enrichissements possibles du modèle

Nous avons finalement abouti à un modèle simple qui répond à nos objectifs initiaux. La prescriptivité du modèle nous contraint à un fonctionnement basique que résume l'algorithme de la figure 1.6 : le moteur fonctionne en régime irrévocable à partir d'un nodule initial jusqu'à ce que le nodule courant ne puisse choisir de nodule successeur (il est alors dit "nodule final").

Cette prescriptivité nous interdit de considérer un troisième facteur dans la prise de décision lors du choix du nodule successeur. Ce troisième facteur, qui interviendrait en plus de l'environnement et du projet en cours, pourrait être une composante liée au hasard, au sentiment, à l'émotion, ou encore à la ré-flexion,.... Cette dernière caractéristique fait référence à la capacité d'anticiper, de raisonner sur ses propres représentations, qui est la marque des véritables "agents cognitifs".

Prendre du recul par rapport à ses connaissances (*se tourner vers soi*, c'est ce que signifie étymologiquement ré-flexion), c'est être capable de simuler son propre raisonnement sans s'y engager. Se dessine ainsi un fonctionnement réflexif du moteur qui consisterait à explorer les différents parcours possibles à partir d'un nodule, préalablement au fonctionnement basique, ou du moins indépendamment de celui-ci. On rejoindrait alors des préoccupations classiques de résolution de problèmes en IA.

Examiner les conséquences d'un choix, avant de choisir véritablement est intéressant lorsqu'on dépasse les limites de la prescriptivité que nous avons prise

comme objectif : un système qui doute de ses connaissances, qui sait que celles-ci ne sont pas toutes fiables, a besoin d'anticiper, de prendre du recul pour éviter de trop se tromper. Mais ce type de système, pour fonctionner, doit avoir accès à d'autres types de connaissances, comme "comment savoir que je me trompe ?"[17]. L'approche que nous avons prise est d'une certaine façon inverse puisque nous nous appuyons sur un oracle, le concepteur, dont l'infaillibilité n'est pas mise en cause, pour corriger les connaissances qui sont détectées erronées ou incomplètes après exécution. C'est là en quelque sorte une approche de type "agir puis réviser" qui pourrait s'opposer à une approche de type "réfléchir avant d'agir", plus élaborée mais nécessaire en cas d'absence d'oracle fiable...

1.4 Discussion et conclusion

1.4.1 Environnement versus Projet

"Plutôt que de s'appuyer sur le raisonnement pour intervenir entre la perception et l'action, nous croyons que l'activité dérive en grande part de sortes de mécanismes très simples interagissant avec la situation immédiate"

[Agre et Chapman, 1987, p. 268]

Le modèle Situation-Action que nous avons exposé dans ce chapitre concilie les deux aspects antagonistes de la cognition :

- la réactivité à l'environnement que recherchent les concepteurs de robots situés comme Philip Agre et David Chapman,

- la planification nécessaire à l'accomplissement de tâches pour lesquelles il est nécessaire de suivre un ordonnancement dans les actions et de mémoriser des informations sur l'environnement.

Cet antagonisme se retrouve dans les architectures des systèmes d'IA. On peut en effet distinguer le contrôle "opportuniste" (par exemple dans le modèle "tableau noir"), du contrôle "impératif", qui suit classiquement une stratégie préétablie ([Causse, 1994]).

L'opposition entre "comportement guidé par l'environnement" et "comportement guidé par le projet de l'agent" apparaît fondamentalement dans notre modèle, où les deux dimensions, environnement et projet, sont constitutives de notre objet central : la situation. Notre modèle permet ainsi de représenter pour chaque situation le choix qui résulte de l'intégration de ces deux composantes

[17] Pour répondre à cette question, les systèmes d'IA basés sur la recherche exploratoire intègrent nécessairement une description explicite du but recherché, ce qui n'est pas possible dans beaucoup de domaines applicatifs de l'IA.

(comment les nouvelles informations renvoyées par l'action courante modifient le projet en cours).

1.4.2 La notion d'action

La notion d'action d'un nodule de situation est proche de celle développée par P. Agre et D. Chapman pour le système Pengi ([Agre et Chapman, 1987]). Ces derniers refusent de maintenir une représentation complète de l'environnement dans Pengi, et ne veulent retenir que les aspects "indexicaux-fonctionnels" d'une situation, c'est-à-dire les propriétés de la situation courante qui :

- concernent directement l'agent et son environnement proche (aspects indexicaux),

- sont pertinentes pour la tâche en cours (aspects fonctionnels).

L'attachement des actions aux situations correspond dans notre modèle au même souci d'exécuter des actions définies spécifiquement pour être utiles dans la situation courante et renvoyer les informations importantes pour le choix de la nouvelle situation.

Notre conception de la notion d'action rejoint aussi celle de Walter Van de Velde de la notion d'*handler*, cette fois-ci à cause du rôle joué par l'action/handler dans la modification / instanciation du modèle de la tâche :

"Les 'handlers' manipulent le modèle de la tâche d'une instance de tâche pour permettre aux méthodes d'atteindre le but du raisonnement. Ils essaient de modifier le modèle de la tâche de manière à ce que les méthodes puissent inférer suffisamment d'informations sur le cas. Ce faisant, ils provoquent des effets de bord sur l'instanciation des sous-tâches. Les 'handlers' peuvent retirer des instanciations de la mémoire, ou mettre en oeuvre une perception ou une action sur le monde. Il n'y a pas de différence entre tout cela ; tout est fait pour obtenir un effet de bord qui permette le raisonnement"

[Van de Velde, 1990, p. 459]

1.4.3 Un premier bilan ?

Incrémentalité, compréhensibilité et révisabilité étaient les trois caractéristiques essentielles que devait faciliter le modèle de représentation de savoir-faire que nous recherchions. En cette fin de premier chapitre, il est prématuré de se prononcer sur la réalisation ou non de ce but par la représentation par nodules de situation. Nous présenterons dans les chapitres suivants des exemples concrets à partir d'outils construits dans le prolongement de ce modèle. Nous discuterons aussi

de la représentation des connaissances et de l'outil de révision, de manière comparative à d'autres représentations des connaissances et outils de révision.

Autour du problème de la compréhensibilité, on pourrait chercher à défendre l'argument de la "validité cognitive ou psychologique du modèle Situation-Action". Cette direction a été prise dans [Poittevin, 1993]. Si effectivement il est démontré que le modèle très simple Situation-Action exposé dans ce chapitre, avec les enrichissements esquissés dans la section 1.3.6, peut avoir un sens en psychologie cognitive, il serait alors possible de fournir des arguments "cognitifs" en faveur de la représentation par nodules de situations. Schématiquement, si nous raisonnons "par situations" il est alors rationnel de proposer une représentation "par situations" et de soutenir que cette représentation a des propriétés de compréhensibilité.

Or, à l'heure actuelle, nous ne disposons pas de ce soutien. On peut même défendre que la simplicité du modèle Situation-Action ne mérite pas de tels travaux de psychologie cognitive. L'évidence du modèle suffirait à elle seule. La notion de situation est une notion centrale que chacun comprend (mais pourquoi justement ?). Le point de vue pragmatique défend que seule la pratique d'une telle représentation permet de la valider. Cet aller-retour sciences cognitives - ingénierie informatique est d'ailleurs l'essence même de ce qu'est la discipline de l'Intelligence Artificielle.

Ainsi des questions concrètes viennent tout de suite à l'esprit : faut-il représenter l'immensité des situations concrètes d'un domaine ? Ce modèle permet-il de généraliser des situations ? Par ailleurs, nous avons cherché à montrer dans ce chapitre comment les nodules de situation ont été construits de manière à répondre aux objectifs définis à la section 1.1 (c'est-à-dire simplicité, uniformité, représentation explicite, séquentialité du raisonnement, précision plutôt que de concision). Cette dernière caractéristique rejoint la préoccupation évoquée à l'instant : si la précision joue au dépend de la concision, ne va-t-elle pas contre notre objectif de compréhensibilité ? Ces questions concrètes nécessitent des expérimentations concrètes, à défaut de réponses immédiates. Le chapitre suivant détaille la mise en oeuvre d'un outil de construction de BC sous forme de nodules de situation. Une méthodologie pour la construction de graphes de situation est proposée également. Nous reviendrons donc sur les questions posées dans cette section à la fin du chapitre suivant.

Chapitre 2 :

Un outil et une méthodologie d'acquisition des connaissances

Le modèle général défini dans le premier chapitre a été appliqué dans le cadre d'un projet de construction d'un serveur télématique pour la S.N.C.F. La confrontation du modèle théorique général à un besoin concret a permis au cours d'une première étape de mettre à l'épreuve et d'enrichir le modèle. Cette première étape a donc abouti à la définition de spécifications pour la réalisation d'un outil de construction de BCs sous forme de nodules de situation. Conformément à ces spécifications, l'outil EDINOS (pour EDIteur de NOdules de Situation) a été développé lors d'une seconde étape.

Dès lors, l'existence d'un outil comme EDINOS, concrètement utilisable et disponible pour tenter diverses expérimentations, entraîne des conséquences intéressantes :

- d'abord parce que seule une pratique de la modélisation par nodules de situation permet de valider l'intérêt du modèle et de la révision incrémentale de BCs exprimées sous forme de nodules de situation.

- ensuite parce que l'outil une fois développé, permet très facilement de construire de nouvelles BCs et d'appliquer ainsi le modèle de représentation des connaissances à des domaines autres que celui des dialogues Minitel (par exemple : diagnostic de pannes, accord du participe passé,...).

- enfin la pratique répétée de la construction de graphes de nodules permet d'élaborer progressivement une méthodologie générale d'acquisition incrémentale de BCs exprimées sous forme de nodules de situation (troisième étape).

Dans ce chapitre, nous commencerons par détailler le cadre applicatif qui a été le nôtre, ainsi que l'outil EDINOS qui a été construit dans ce contexte. Nous présenterons ensuite notre méthodologie empirique de construction incrémentale de graphes de nodules de situation, et discuterons de l'intérêt de ce type d'approche pour l'acquisition des connaissances, après avoir tiré quelques conclusions sur l'application de l'outil et dégagé des perspectives de développement d'EDINOS.

2.1. L'application aux services télématiques

2.1.1 Le cadre du projet

Le projet des Caisses de Prévoyance et de Retraite de la S.N.C.F. consiste à réaliser un serveur télématique à destination des agents S.N.C.F. et de leurs ayants droit.

Nous distinguons trois acteurs dans le projet :

- l'expert, spécialiste du domaine de l'Assurance Maladie ou de l'Assurance Vieillesse,

- le concepteur de la BC, qui définit, en lien avec l'expert, le contenu des écrans Minitel et le déroulement du dialogue, qui prend la forme de questions / réponses. C'est lui qui est l'utilisateur de l'outil EDINOS de construction des dialogues Minitel.

- le développeur, informaticien en charge du développement de programmes spécifiques, à la demande du concepteur.

Mentionnons également le minitéliste, affilié des Caisses de Prévoyance et de Retraite (C.P.R.) de la S.N.C.F., qui sera l'utilisateur final du serveur télématique et qui donc peut être vu comme le “client” principal de ce projet. Par ailleurs, il convient de bien distinguer l'outil de construction et de simulation des dialogues Minitel, du service Minitel final exploitant la BC définie par l'outil de construction. Le premier doit fonctionner sur le poste de travail du concepteur, le second est en place sur un serveur télématique connecté au réseau TRANSPAC.

Le serveur Minitel est appelé à remplir quatre types de services :

a) permettre aux affiliés d'effectuer des demandes de documents et de formulaires,

b) fournir aux affiliés des informations générales ou personnalisées, relatives à l'Assurance Maladie et à l'Assurance Vieillesse,

c) permettre aux affiliés de poser des questions via un système de messagerie puis de consulter les réponses données par les spécialistes,

d) permettre aux affiliés de consulter directement certaines données propres à leur dossier.

Fournir des services du type b), et plus précisément construire des dialogues personnalisés, n'est pas une tâche facile. Par “dialogue personnalisé”, il faut comprendre “dialogue adapté au profil du minitéliste”. Une caractéristique notable du projet est que le minitéliste n'est pas anonyme, puisqu'au début de la session Minitel il lui est demandé de taper son numéro d'immatriculation S.N.C.F.. Cette saisie a pour but non seulement de faire en sorte que ce service Minitel ne soit utilisé que par les bénéficiaires des C.P.R., mais aussi de permettre d'identifier précisément le minitéliste, afin d'être ensuite en mesure d'utiliser les informations stockées dans les Bases de Données des C.P.R. pour proposer un dialogue le plus concis et adapté possible. Disposer d'informations sur le minitéliste rend ainsi possible la construction de dialogues dits “personnalisés”.

Dans les services du type b), on distinguera donc :

- la mise à disposition d'informations **générales**, par exemple “Les dernières nouvelles de la Caisse de Prévoyance”. Ces informations peuvent être consultées par tout le monde, c'est-à-dire par tous les utilisateurs du service télématique (affiliés des C.P.R.), même si elles ne concernent pas toujours directement tout le monde.

- la mise à disposition d'informations **personnalisées**. Dans ce cas, les dialogues proposés au minitéliste sont adaptés aux catégories dans lesquelles on peut le classer. On évite ainsi de proposer au minitéliste des informations qui ne le concernent pas, et l'on peut focaliser le dialogue (plus concrètement les propositions dans les menus) sur les informations pertinentes pour son cas particulier. Ce type de service peut être d'un grand confort d'usage et d'une grande utilité pour le minitéliste qui, se laissant complètement guider au fur et à mesure du déroulement du dialogue, aboutit à l'information recherchée sans être noyé dans la masse de l'ensemble des informations disponibles. Un des projets des C.P.R. est de mettre à disposition de cette manière le contenu du guide de l'affilié et du guide du retraité, qui sont diffusés par ailleurs sous la forme de guide papier. En plus de l'aspect

convivial, les avantages d'un guide “télématique” personnalisé bien conçu sont les suivants :

- diminution du nombre d'appels téléphoniques pour des renseignements,
- actualisation du contenu du guide plus réactive et moins coûteuse que sous forme papier.

2.1.2 Adéquation avec notre modèle

C'est pour le problème de construction de dialogues personnalisés, que l'application de notre modèle de représentation des connaissances par nodules de situation va s'avérer utile.

Une des spécificités de ce type de dialogues personnalisés tient au fait que les informations nécessaires à l'orientation du dialogue ne sont pas toutes connues au commencement de la session. Elles sont acquises en cours de session, soit par des questions posées directement au minitéliste, soit par la recherche des informations sur le minitéliste dans des Bases de Données “Affiliés”. Comme toutes les informations qui peuvent être utiles à la construction d'un dialogue personnalisé ne sont pas contenues dans les Bases de Données, il est nécessaire de poser des questions au minitéliste. Par ailleurs, il existe plusieurs Bases de Données distinctes et leur accès en temps réel n'est jamais instantané : il est important de s'attacher à faire uniquement les requêtes utiles au dialogue courant et d'aller donc chercher les informations de manière progressive.

Une deuxième caractéristique du projet est le fait que la BC qui définit l'enchaînement des écrans Minitel est par nature sujette à révision, d'abord parce que les informations à diffuser varient dans le temps, mais également parce que le dialogue prévu et défini dans la BC est rarement définitivement validé. En effet, s'il est important que les informations données soient elles-mêmes valides dans leur contexte d'affichage, l'ordre des questions posées au minitéliste afin de cerner son besoin et sa catégorie n'est pas toujours optimal. Ce type de connaissances (par exemple, éviter de poser telle question inutile dans tel contexte) n'est pas essentiel au fonctionnement du serveur et celui-ci peut fonctionner sans que la BC ne fournisse un dialogue optimisé pour chaque catégorie de minitélistes. Ceci est d'autant plus vrai que la population de certaines catégories de minitélistes est extrêmement réduite et la définition de dialogues personnalisés optimaux pour ces quelques cas particuliers est un investissement qui n'est pas considéré comme rentable. L'approche générale adoptée pour ce type de dialogues particuliers est de prendre en

compte et corriger les imperfections au fur et à mesure de leur détection et de l'évaluation de leur importance.

En résumé, l'analyse des besoins du projet "Serveur télématique à destination des affiliés" fait ressortir les deux caractéristiques principales suivantes :

a) la nécessité d'aller chercher les informations sur le minitéliste progressivement au cours du dialogue, et uniquement lorsque ces informations sont utiles,

b) la nécessité de pouvoir facilement et régulièrement corriger et enrichir la BC qui définit l'enchaînement du dialogue.

Pour ces deux raisons, il paraît intéressant d'utiliser les nodules de situation pour représenter les connaissances qui définissent l'organisation du dialogue Minitel. Alors que la caractéristique b) correspond à l'objectif général défini dans le chapitre 1 (révisabilité), la caractéristique a) exige une certaine lisibilité des différentes lignes de dialogues possibles. Or, pour permettre au concepteur de suivre dans la BC la chronologie de l'enchaînement des écrans, il faut offrir une représentation explicite et séquentielle du raisonnement : cela fait partie des objectifs des graphes de nodules de situations.

2.1.3 Les adaptations du modèle

Pour être appliquée à la conception de dialogues de serveur Minitel, la représentation par nodules de situation ébauchée dans le chapitre 1 doit être adaptée. Le modèle initial, basique et très général, est légèrement modifié ou élargi pour permettre de répondre à certaines contraintes, propres au projet ou non. La conséquence de ces adaptations du modèle aboutissent à la définition d'un modèle à la fois plus riche et plus précis que le modèle proposé dans le chapitre 1.
Nous distinguerons deux types d'adaptations :

- les **instanciations** sont les adaptations du modèle qui sont spécifiques à l'application. Ces instanciations sont des spécialisations du modèle initial, qui ont un intérêt limité en terme de réutilisation et n'aboutissent donc pas à des propositions de modification du modèle général de représentation par nodules de situations.

- les **extensions** complètent le modèle afin d'aider davantage à réaliser les objectifs fixés dans le chapitre 1 : l'application du modèle initial au projet de la S.N.C.F. fait ressortir certains manques à combler pour que le modèle gagne en intérêt et en généricité.

2.1.3.1 Les instanciations

Les instanciations sont des adaptations particulières de notre modèle à ce projet de construction de dialogues Minitel.

L'idée centrale de l'application de la représentation par nodules de situation à la construction de dialogues télématiques est que les branches des dialogues correspondent à l'exécution de graphes de nodules. Dans ces circonstances, les actions des nodules sont des questions posées au minitéliste.

La principale instanciation de notre modèle est donc la création d'une classe d'objets particuliers, les **écrans**, pour permettre au concepteur de complètement définir les types d'actions particuliers que sont les questions posées au minitéliste.

D'autres adaptations doivent aussi être introduites pour prendre en compte la pression sur les touches SOMMAIRE, SUITE et RETOUR au cours du dialogue, de manière à ce que les conventions de navigation Minitel soient respectées.

a) Les notions de procédure et d'écran

Nous allons dans cette section spécifier deux types d'objets différents qui correspondent à deux types d'actions différents.

Dans le cadre du projet “Serveur télématique à destination des affiliés”, trois types d'actions, au sens du modèle Situation-Action, doivent être nécessairement effectuées par le SBC :

1) la recherche d'information dans des Bases de Données,

2) la question posée directement au minitéliste (question à choix multiple),

3) le message final donné au minitéliste : la conclusion affichée en fin de dialogue.

Les actions de type 1 peuvent être définies sur la base d'une collaboration entre les deux acteurs du projet que sont l'informaticien et le concepteur du dialogue. La notion de procédure est introduite au sein du modèle pour permettre de préciser ce type d'action. Une **procédure** est définie comme une action automatisée qui, même si elle n'est pas réalisée dans l'outil de construction de la BC, le sera dans le système final. Le concepteur et l'informaticien font ensemble l'analyse de la procédure, en spécifiant ce que doit faire la procédure et en définissant la forme des données attendues en sortie. Cette déclaration des faits que peut envoyer la procédure est appelée *définition* de la procédure (cf. section 1.3.5.3). La définition de la procédure est un peu *l'interface* au sens de la programmation (objet) de la procédure dans la BC : c'est sur la base de sa définition que le concepteur fait appel

à la procédure depuis des nodules de situation. Le système de construction et de simulation de la BC est capable de simuler l'exécution de la procédure, à partir de cette seule définition. L'informaticien est en charge de programmer la procédure, de la valider et d'assurer son intégration au sein du système final.

Les actions de type 2 et 3 ont en commun l'affichage d'un écran au minitéliste. Y compris dans le cas d'un message final, on attend de la part du minitéliste une réponse à la suite de l'affichage de l'écran : par exemple qu'il tape sur son clavier le chiffre '1' puis la touche 'ENVOI', ou qu'il tape simplement la touche 'SOMMAIRE'. Le contenu de l'écran affiché, ainsi que les réponses du minitéliste permises et attendues, doivent pouvoir être spécifiées par le concepteur. La notion d'*écran* est donc proposée pour définir, au sein de l'outil de construction des dialogues, ce type d'action qui consiste à afficher une information au minitéliste, puis à contrôler et à interpréter les réponses données. Un écran est en quelque sorte un type particulier de procédure, que le concepteur peut définir sans l'aide de l'informaticien. Un écran contient le contenu du texte à afficher, ainsi que les règles de traduction des réponses de l'utilisateur en faits qui seront stockés dans la Mémoire de Travail. Ces règles de traduction sont assez simples dans le cadre de notre projet où les questions sont à choix multiple.

Le concepteur du dialogue a ainsi la possibilité de définir complètement certains types d'actions (les écrans). En ce qui concerne les actions qu'il ne peut développer tout seul (les procédures), il reste maître de leur spécification.

Nous distinguerons en fin de compte les actions qui consistent à exécuter une procédure, qu'on appellera *actions "évaluer"*, des actions qui consistent à afficher un écran, puis interpréter les réponses du minitéliste, qu'on appellera *actions "demander"*.

b) La navigation

- la touche SOMMAIRE

L'application du modèle Situation-Action aux dialogues Minitel soulève un problème qu'on peut plus généralement lier à la navigation de type hypertexte. Dans le modèle Situation-Action, on part d'une situation initiale et on aboutit à une situation finale tout en exécutant un certain nombre d'actions. Ce type de fonctionnement est dit irrévocable parce que chaque pas effectué l'est de manière définitive. Or cette interdiction du retour en arrière n'est pas compatible avec l'utilisation courante du Minitel où la touche SOMMAIRE est très communément employée et provoque un retour au dernier menu affiché.

Une solution consiste à considérer qu'un appui sur la touche SOMMAIRE provoque la terminaison du moteur en cours d'exécution et lance un nouveau moteur qui virtuellement parcourt à nouveau le graphe depuis le nodule initial jusqu'au nodule correspondant au dernier menu, appelé nodule "sommaire". Concrètement le moteur doit se retrouver dans le même état (la même MT) que lorsque le nodule "sommaire" a été activé une première fois au cours du premier lancement du moteur.

Nous avons choisi de permettre l'attachement à tout nodule d'une propriété "sommaire", dont la signification est la suivante : dès lors que ce nodule a été activé, toute pression sur la touche SOMMAIRE provoquera, quel que soit le nodule courant, le retour à ce nodule avec l'état présent de la MT. Cette propriété "sommaire" a des conséquences sur le fonctionnement du moteur : c'est donc bien une propriété du nodule, non de l'action (écran). L'introduction de cette propriété nécessite le stockage dans l'historique des nodules activés des états correspondants de la MT.

Ce problème de sauvegarde de contexte pour permettre de véritables retours en arrière est un problème commun à toutes les navigations de type hypertexte.

- les touches SUITE et RETOUR

Si l'on suit les recommandations TELETEL pour l'utilisation des touches de fonction Minitel, la touche SUITE doit permettre l'accès à la suite du texte lorsque celui-ci est partitionné en plusieurs écrans. Pour des raisons ergonomiques évidentes, la touche inverse RETOUR doit permettre de rendre réversible l'opération.

Il est nécessaire de bien faire la distinction entre les opérations de choix parmi un ensemble d'alternatives possibles, et les opérations de chaînage d'écrans

"liés". Le premier type d'opération doit être représenté au niveau du nodule de situation : les différentes alternatives possibles sont autant de faits qui peuvent être renvoyés et qui servent au choix du nodule suivant. Ces opérations de choix sont effectuées par le minitéliste à l'aide de la touche ENVOI et rendues réversibles par la touche SOMMAIRE. Le second type d'opération est conceptuellement plutôt du "niveau action", puisqu'il s'agit du simple affichage d'un texte partitionné en plusieurs écrans. Nous faisons l'hypothèse que le minitéliste utilise les touches SUITE et RETOUR pour "naviguer" dans ce texte, sans que cela ne produise d'information utile à la suite du dialogue.

La notion d'écran doit donc être enrichie de manière à permettre au concepteur d'exprimer ces chaînages d'écrans. L'appel à un ensemble d'écrans "liés" doit correspondre à une seule action.

2.1.3.2 Les extensions

Les extensions sont des adaptations du modèle moins anecdotiques que celles exposées dans la section précédente, parce qu'elles sont des enrichissements du modèle qui se sont révélés assez vite nécessaires et dépassent le cadre de l'application. Ces extensions concernent le langage de la Mémoire de Travail (MT) et deux nouveaux types d'action : les actions "déclencher" qui permettent d'appeler des sous-graphes, et les actions "déduire" qui permettent d'utiliser des Bases de Règles.

a) Le langage de la MT

Représenter les faits de manière propositionnelle permet au concepteur de tout exprimer à condition de définir autant d'actions que nécessaire. Par exemple, il peut décider de créer une action *âgeDeAffilié1* qui renvoie *affiliéAMoinsDe25ans* ou *affiliéAPlusDe25ans*. Mais il peut aussi avoir besoin plus loin dans le dialogue d'exécuter une action *âgeDeAffilié2* qui renvoie trois faits : *affiliéAMoinsDe50ans*, *affiliéAEntre50Et55ans* ou *affiliéAPlusDe55ans*.

Introduire la notion d'**attribut** et de **valeur** permet d'enrichir la puissance d'expression des faits de la MT. On pourra ainsi définir une seule action *âgeDeAffilié* qui retourne pour l'attribut *âgeDeAffilié* une valeur entière (par exemple *52*).

L'introduction des notions d'**objet** et de **relation entre objets** poursuit ce processus d'enrichissement du langage de la MT pour diminuer le nombre d'actions

à définir. Une action *âge* appliquée à l'objet *affilié* peut alors renvoyer la valeur *52* pour l'attribut *âge* de l'objet *affilié*. La même action *âge* peut être appliquée à l'objet *conjoint* : il est inutile de définir une nouvelle action spécifique pour rechercher l'âge du conjoint.

De même, supposons que l'on ait besoin de connaître l'âge de l'ouvrant-droit du conjoint, alors que l'on connaît déjà par ailleurs l'âge de l'affilié en question (attribut *âge* de l'objet *affilié*). L'établissement d'une relation *ouvrantDroit* de l'objet *conjoint* vers l'objet *affilié*, permet d'éviter la création d'une action *âgeDeOuvrantDroitDuConjoint*. Il suffirait en effet directement d'aller chercher la valeur de l'attribut *âge* de l'objet qui est lié à l'objet *conjoint* par la relation *ouvrantDroit*.

Pour prendre en compte ces différents enrichissements dans l'outil de construction et de simulation de la BC, il faut :

- enrichir les langages utilisés par le concepteur pour définir les actions, les faits, les règles.

- développer des programmes spécifiques pour stocker dans la MT les faits renvoyés par les procédures, puis pour rechercher une information au sein de la MT.

En effet, la représentation propositionnelle permettait d'utiliser un même langage pour exprimer :

- les faits que renvoient les procédures au cours de l'exécution d'un graphe,
- le contenu de la MT au cours de l'exécution d'un graphe (liste de faits),
- les prémisses des règles de choix des nodules.

Tel n'est plus le cas si l'on veut prendre en compte les enrichissements proposés ci-dessus :

- **le langage de définition des actions** doit être modifié. Le concepteur doit pouvoir définir :

  - les *objets* que l'action est susceptible de renvoyer,

  - les *attributs* que ces objets sont susceptibles de posséder, ainsi que le *type* de ces attributs (nous définirons dans un premier temps trois *types* d'attributs : CHAÎNE, ENTIER, RÉEL).

  - les *relations que* ces objets sont susceptibles de posséder avec d'autres objets déjà définis.

L'annexe 1 contient la grammaire de ce langage.

- **un langage des "entrées de la MT"** doit être spécifié pour définir concrètement les faits qui devront être "stockés" dans la MT durant l'exécution du graphe. On utilisera le terme **fait** pour désigner une information élémentaire

renvoyée par la procédure durant l'exécution du graphe. Par exemple : la procédure *conjoint* renvoie les faits "**conjoint1.âge 45**" et "**conjoint1.ouvrantDroit affilié1**" (création du couple attribut-valeur "âge 45" et établissement de la relation *ouvrantDroit* avec l'objet *affilié1*).

- **un langage de présentation du contenu de la MT** doit être spécifié. Ce langage n'est pas forcément le même que celui des entrées de la MT, même si ce dernier peut suffire à connaître "ce que contient effectivement la MT". Plutôt que la liste chronologique des faits renvoyés par les actions exécutées, une présentation graphique des objets, avec leurs attributs et relations, est préférable.

- **le langage des actions des nodules** doit être modifié afin de permettre au concepteur de préciser éventuellement un ou plusieurs objets sur lesquels, ou autour desquels, l'action doit être exécutée. Par exemple, "évaluer la procédure *âge* sur l'objet *affilié1"*.

- **le langage des prémisses des règles de choix des nodules** doit être enrichi pour permettre d'exprimer des propositions complexes. On définira le terme *requête* pour qualifier la recherche d'une information dans la MT.

Exemple de requête : "conjoint.ouvrantDroit.âge"

Exemple de proposition : "conjoint.ouvrantDroit.âge = 52"

Exemple de prémisse : "(conjoint.ouvrantDroit.âge = 52) ET retraité"

Une requête peut renvoyer cinq types de valeurs : la valeur **vrai**, un entier, une chaîne, un réel, la valeur **nil** (en cas d'échec de la requête, si par exemple il est fait référence à un objet, une relation, un attribut qui n'est pas connu dans la MT).

On utilisera la règle suivante pour l'évaluation des prémisses: si la requête renvoie nil, la proposition qui contient la requête est fausse. Cette règle est la conséquence de l'hypothèse du monde fermé retenue dans le chapitre 1 : tout fait qui n'est pas présent dans la Mémoire de Travail au moment où celle-ci est consultée (i.e. lors du choix du nodule suivant) est considéré comme faux. Autrement dit, à l'instant t, n'est considéré comme vrai que ce que contient la MT.

La deuxième extension apportée à notre modèle est une extension fondamentale parce qu'elle permet de réutiliser les graphes déjà créés : il s'agit des actions de type "déclencher".

b) L'action “déclencher”

La modularité d'une représentation peut être définie par sa capacité à offrir des composants **réutilisables**. La modularité est une qualité incontournable pour l'acquisition de connaissances. A partir du modèle défini dans le chapitre 1, deux types de modularité sont possibles, à des niveaux de granularité différents :

- première option : on peut rechercher la modularité au niveau du *nodule*,
- deuxième option : rechercher une modularité au niveau du *graphe*.

Nous avions précisé qu'un nodule appartenait à un graphe, le graphe représentant une tâche. La première option consiste à permettre à un nodule d'appartenir à plusieurs graphes. Le nodule est alors modulaire, puisqu'il peut être (ré)utilisé pour accomplir plusieurs tâches différentes. L'inconvénient majeur de cette option est dans la compréhensibilité du nodule. Comme nous le détaillerons dans la section 2.3, un nodule peut avoir un niveau d'abstraction plus ou moins élevé. Plus un nodule est abstrait, c'est-à-dire plus il représente un grand nombre de situations concrètes différentes, plus il risque d'être complexe (en terme de règles de choix, de nombre de nodules successeurs possibles,...), moins il est facilement compréhensible. Choisir d'avoir des nodules globalement modulaires, donc permettre à tout graphe de faire appel à tout nodule, minimise le nombre total de nodules, mais incite à une forte abstraction conceptuelle des nodules qui se paie en perte de compréhensibilité

La seconde option offre un bon compromis entre modularité et compréhensibilité. Elle consiste à s'en tenir au principe suivant :

Un nodule n'appartient qu'à un et un seul graphe.

Principe de modularité médiane

Ce principe de modularité médiane doit être accompagné d'un enrichissement des types d'actions possibles. Il faut en effet pouvoir, depuis n'importe quel nodule, lancer un autre graphe. On donne alors à l'objet *graphe* une modularité : un graphe accomplissant une tâche peut être appelé par un autre graphe. C'est l'action dite “déclencher” qui permettra cela. Elle consiste à provoquer, depuis le nodule courant du graphe en cours d'exécution, l'exécution d'un sous-graphe. Une fois que ce sous-graphe a été exécuté (après l'exécution du nodule final du sous-graphe), le moteur revient au nodule “appelant” avec la MT enrichie par l'exécution du sous-graphe.

Dans ces conditions, un nodule ne prend place qu'à l'intérieur d'un seul graphe. Il prend son sens dans ce graphe, et peut être compris par sa place dans ce

graphe. L'unité modulaire est alors le graphe ou la tâche. A la suite de la décomposition d'une tâche en sous-tâches, il devient possible de réutiliser des sous-tâches déjà créées.

c) L'action “déduire”

Le système exploitant une BC définie sous la forme de nodules de situations n'est pas doté des capacités déductives des systèmes experts à base de règles. Cela n’est pas une limitation dans la mesure où nous considérons que représenter précisément et explicitement des savoir-faire ne nécessite pas de capacités déductives. Pourtant il serait intéressant de permettre d'intégrer des connaissances déjà construites sous forme de base de règles dans un SBC fonctionnant avec des nodules de situation.

Comment fonctionne un système déductif ? A partir d'un ensemble de faits (c’est-à-dire à partir d'une MT) un système déductif est capable de déduire de nouveaux faits en exploitant une base de règles. Dans l'optique du schéma Situation-Action, ce type d'inférence peut être effectué par une action. On peut donc proposer de créer un quatrième type d'action (après l'action “évaluer” pour les procédures, l'action “demander” pour les écrans, et l'action “déclencher” pour les graphes) : l'action “déduire”, qui à partir de l'état de la MT, utilise une base de règles pour chercher à effectuer *certaines* déductions. En effet, le principe de “perception active” défini dans la section 1.3.5.1, nous contraint à préciser à l'avance les faits que l'on recherche lorsque l'on fait appel à une action “déduire”. Par exemple, on peut chercher à déduire la valeur de l'attribut *antenne* de l'objet *affilié1*, avec une base de règles qui contient des règles du type “SI (affilié.département = 13) ou (affilié.département = 83) ALORS affilié.antenne = 'Marseille'”. Les actions de type “déduire” doivent donc, comme les procédures, être *définies*, au sens où les faits qui sont attendus en sortie doivent être *déclarés*.

Très concrètement, une action “déduire” est équivalente à l'appel d'un sous-graphe spécialisé dans la recherche de l'inférence spécifique. Ainsi pour l'exemple des antennes, un sous-graphe contenant un nodule dont les règles de choix orientent sur le nodule correspondant à l'antenne pourrait tout aussi bien être utilisé.
Mais le recours à une base de règles est plus intéressant lorsque les nodules du sous-graphe équivalent ne comportent aucune action, aucune communication avec l'environnement. Lorsque les connaissances à représenter ont un aspect uniquement déclaratif, qu’il s’agit uniquement d’un enrichissement de la MT (une déduction), la règle est un objet plus concis, plus facile et plus rapide à créer. C'est le cas pour l'exemple de l'antenne tel qu'il a été donné : la tâche est simplement d'ajouter un

attribut *antenne* à l'objet *affilié1*. Par contre si le but avait été de fournir l'adresse de l'antenne en question, le problème aurait été différent : la représentation par nodules de situation permet plus facilement de spécifier la marche à suivre dès lors que celle-ci est tributaire des résultats de l'exécution d'une action.

Il serait intéressant de délimiter de manière plus générale la frontière entre "connaissances qu'il vaut mieux représenter sous la forme de règle" et "connaissances qu'il vaut mieux représenter sous la forme de nodule de situation"... Nous avons déjà évoqué dans l'introduction l'opposition entre "connaissances opératoires" et "connaissances substantives". Cette opposition est la même que celle qu'établit Tom Gruber entre "connaissances stratégiques" et "connaissances substantives"[1] :

> *"Les **connaissances stratégiques** sont les connaissances utilisées par un agent pour décider quelle action doit être effectuée ensuite, au sens où les actions ont des conséquences extérieures à l'agent. (.../...) Le terme plus général '**connaissances de contrôle**' fait référence aux connaissances utilisées pour décider ce qui doit être fait ensuite"*
>
> [Gruber, 1989b, p. 5]

T. Gruber distingue donc les connaissances stratégiques des connaissances de contrôle en spécifiant que les premières sont au niveau "Connaissances", tandis que les secondes sont au niveau "implémentation". Nous ne faisons pas cette distinction niveau implémentation / niveau connaissances, et nous préférons utiliser le terme générique de connaissances **opératoires**. Un exemple de connaissances opératoires serait la réponse à la question suivante : "comment savoir si un affilié a droit à un examen de santé ?".

Par contre, nous retenons exactement le terme et la définition donnée par T. Gruber pour qualifier les connaissances qu'il appelle **substantives** et qui représentent "ce qu'un agent croit être vrai dans le monde". Ce sont des connaissances générales, qui portent sur des états de choses. Un exemple de connaissances substantives est la proposition suivante : "les affiliés retraités ont droit à un examen de santé tous les cinq ans". Comme le montrent les deux exemples choisis ci-dessus, la frontière n'est pas forcément très nette entre connaissances substantives et connaissances opératoires. Cela peut s'expliquer par l'hypothèse que les connaissances substantives sont des connaissances opératoires abstraites (ou décontextualisées).

Nous avons vu qu'il pouvait y avoir une certaine équivalence entre une base de règles et un graphe. Pour cette raison, le schéma suivant qu'on pourrait

---

[1] C'est d'ailleurs à T. Gruber que nous empruntons le terme "connaissances substantives".

proposer :*"les connaissances opératoires doivent être exprimées sous forme de nodules, les connaissances substantives doivent être exprimées dans une base de règles"* possède ses limites, qui sont liées à ce que, comme nous venons de le souligner, la frontière entre les deux types de connaissances n'est pas rigide.

2.2 Un outil basé sur les nodules de situation : EDINOS

A partir du modèle et des adaptations que nous venons de passer en revue, nous avons développé un environnement de construction de Bases de Connaissances exprimées sous la forme de nodules de situation. Cet outil, baptisé EDINOS pour EDIteur de NOdules de Situation doit permettre au concepteur de construire, modifier et simuler les dialogues Minitel.

Nous présentons d'abord les spécifications de cet outil, qui sont une proposition d'implémentation du modèle défini jusqu'ici, dans un contexte et une organisation de travail particuliers. Puis nous illustrerons certains détails de la réalisation finale, soit pour montrer de quelle manière les spécifications ont été réalisées, soit pour préciser certains choix qui ont été faits et qui, pour certains d'entre eux sont des écarts par rapport aux spécifications initiales. Notre travail nous a en effet amené à préciser des aspects qui ne relèvent pas directement d'un travail de recherche, mais qui ont rendu celui-ci "expérimentable"...

2.2.1 Les spécifications

2.2.1.1 Des caractéristiques générales

Nous allons développer d'abord trois caractéristiques générales qu'EDINOS doit remplir.

La première caractéristique est un principe général qu'on voudrait voir réalisé au sein d'EDINOS. Ce principe, appelé "compréhension par exploration", consiste à donner les moyens au concepteur d'explorer la BC à partir des liens définis entre les objets de la BC (nodules, actions, faits).

La seconde caractéristique porte sur l'organisation de travail au sein de laquelle EDINOS sera utilisé et aboutira à la définition de la notion de *base* de nodules.

La troisième caractéristique, liée à la "contrainte d'interruptibilité" définie dans la section 1.1.4, concerne le module de simulation.

a) La "compréhension par exploration"

L'environnement de construction de la BC contient plusieurs types d'objets : des nodules, des actions, des faits. Or ces objets sont liés entre eux : un nodule par exemple est lié à ses nodules ancêtres, à ses nodules enfants, à son action, aux faits que l'action peut renvoyer. La compréhension d'un objet passe par la compréhension de tous les objets qui lui sont liés. Notre point de vue est que par conséquent l'environnement doit faciliter au mieux l'exploration des objets de la BC par l'utilisateur. C'est par cette exploration que l'utilisateur pourra vraiment comprendre un objet O : l'utilisateur doit pouvoir connaître et explorer les objets auxquels l'objet O fait référence, ainsi que les objets qui font référence à l'objet O. Autrement dit lorsque les commentaires associés à l'objet sont peu clairs et ne suffisent pas à l'utilisateur pour comprendre l'objet, celui-ci veut pouvoir chercher dans la BC :

- à quoi est sensé servir l'objet (comment il est utilisé par les autres objets),
- et ce que l'objet fait proprement dit (comment il utilise les autres objets).

Cette idée d'exploration *active* des objets de l'environnement est liée à la notion d'identificateur d'objet que nous avons déjà mentionné, dans la section 1.2.2.1 lorsque nous présentions les environnements de programmation orientée objet. En fin de compte, notre environnement doit contenir les fonctionnalités suivantes :

- un objet possède un identifiant, et il est instantanément accessible à partir de cet identifiant. Ainsi, si l'utilisateur clique sur un identifiant, une fenêtre s'ouvre affichant le "contenu" de l'objet (les champs qui le définissent).
- un outil de recherche permet d'afficher tous les objets faisant référence à un objet donné.

b) L'organisation de travail

Deux types de Bases de Connaissances sont distingués. La BC dite *d'exploitation* est utilisée par le serveur télématique pour afficher les écrans Minitel. Des BCs dites *de test* sont construites et développées par le concepteur des dialogues. Les parties des BCs de test qui ont été validées doivent pouvoir être transférées dans la BC d'exploitation. Les BCs de test doivent pouvoir fonctionner et être modifiées de manière indépendante et isolée du réseau interne de l'entreprise.

L'environnement EDINOS doit donc pouvoir tourner sur des micro-ordinateurs déconnectés du réseau, ainsi que sur certaines parties des BC de tests.

Pour permettre cette modularité de la BC, nous avons introduit la notion de *base*. Une base est constituée d'un ensemble de graphes (donc de nodules), d'un ensemble de procédures et d'un ensemble d'écrans. Une base peut être décrite sous la forme de fichiers texte, qui pourront être transférés via le réseau interne de l'entreprise. EDINOS doit être capable de charger une base à partir de ces fichiers texte, et inversement de sauvegarder une base dans le format de ces fichiers. Ce découpage de la BC en plusieurs bases distinctes nécessite l'ajout d'un certain nombre de contrôles lors du chargement d'une base dans EDINOS, mais aussi lors du transfert d'une base vers la BC d'exploitation. En effet, un objet doit avoir un identifiant unique au sein de la BC toute entière, mais EDINOS ne peut vérifier cette unicité qu'au sein des bases chargées (voir figure 2.1). L'outil de transfert d'une base vers la BC d'exploitation doit donc obligatoirement effectuer un contrôle strict sur les noms des nouveaux objets à insérer dans la BC.

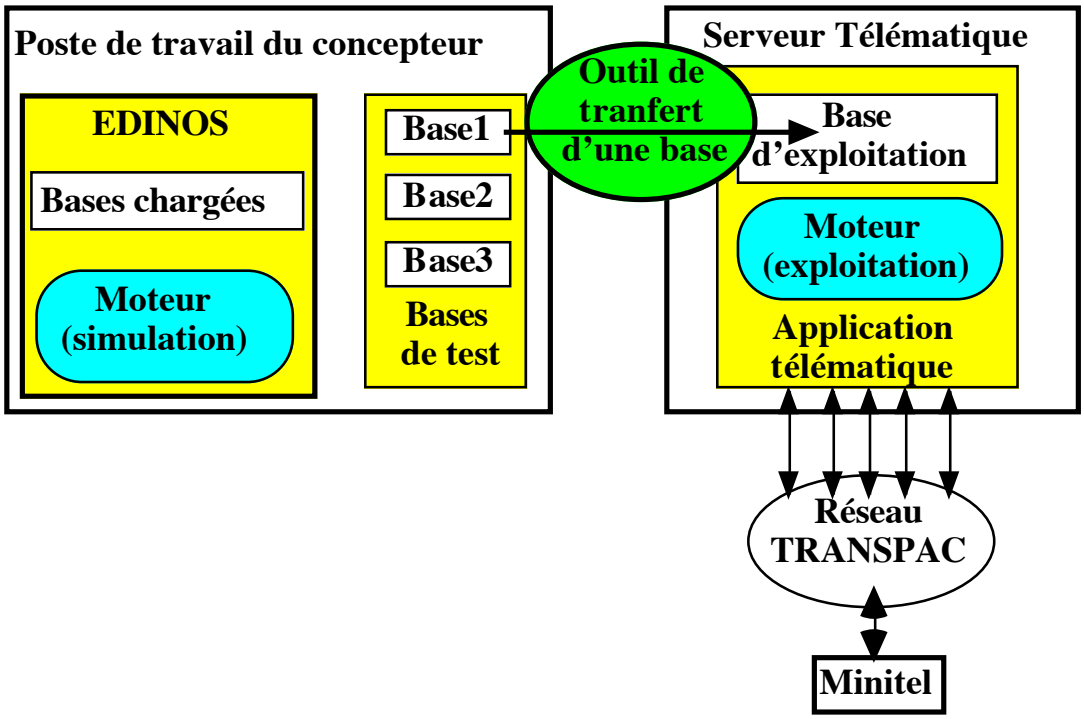


Figure 2.1 : Poste de travail du concepteur et Serveur Télématique

Certains objets peuvent être communs à plusieurs bases. Par exemple une action peut être appelée par des graphes appartenant à des bases différentes. Ces objets "communs" sont regroupés au sein d'une base unique, dite base "commune", qui est chargée automatiquement à chaque lancement d'EDINOS.

Les modifications des objets de la base "commune" doivent être diffusées auprès de tous les utilisateurs d'EDINOS. C'est là la seule contrainte qui lie les différents utilisateurs d'EDINOS. En effet, chacun peut créer et maintenir ses propres bases de test de manière indépendante. Le risque que deux utilisateurs donnent un même nom à un objet est maîtrisé par le contrôle effectué lors du transfert d'une base vers la BC d'exploitation : si le cas se produit, le nom de l'objet "déjà connu" est affiché, le transfert est annulé et l'utilisateur devra définir un autre nom pour l'objet en cause.

c) Le module de simulation

Au sein de l'outil de construction de graphes de nodules de situation, une composante joue un rôle central : il s'agit du module de simulation. Nous avons défini dans la section 1.1.4 une "contrainte d'interruptibilité" du système en fonctionnement. S'il n'est pas intéressant que cette contrainte puisse être vérifiée pour le moteur en exploitation, elle doit pouvoir être remplie au sein de l'environnement EDINOS. Un outil de simulation intégré à EDINOS doit par conséquent permettre au concepteur de tester et valider les dialogues construits. Le concepteur a la possibilité durant toute simulation d'interrompre le moteur et d'examiner "la situation".

Le module de simulation a donc deux fonctionnalités principales :

- il doit permettre au concepteur de comprendre la "situation courante", objet dont nous reparlerons dans la section 2.3.1. Pour cela, il doit pouvoir notamment visualiser l'état de la MT. Les principes de la "compréhension par exploration" exposés ci-dessus doivent être en vigueur.

- le module de simulation doit être capable de simuler les actions effectuées par les nodules (les procédures et écrans introduits dans la section 2.1.3.1). La simulation des écrans consiste à construire un dialogue Minitel similaire à celui qui sera proposé au minitéliste. La simulation des procédures est également nécessaire parce que les procédures ne sont programmées qu'au sein du serveur télématique. Lorsque l'outil de simulation doit exécuter une action "évaluer", il propose à l'utilisateur / concepteur de choisir un fait parmi l'ensemble des faits que peut renvoyer la procédure. C'est la *définition* de la procédure (voir section 1.3.5.3) qui permet de connaître les faits attendus en retour de l'exécution de la procédure.

Nous allons à présent passer en revue les différents objets de l'environnement EDINOS (nodules, écrans, procédures, Mémoire de Travail,...), en précisant à chaque fois comment ceux-ci doivent être représentés au sein d'EDINOS.

2.2.1.2 L'objet "nodule de situation"

En plus des champs "actions" et "règles de choix" qui définissent le comportement du nodule, un nodule de situation contient deux champs supplémentaires :

- un champ "identifiant", qui contient le nom du nodule. Lorsque nous présenterons une méthodologie de construction de graphes de nodules de situation, nous proposerons une convention pour nommer les nodules (voir section 2.3.3).

- un champ "description" permet au concepteur de détailler la signification du nodule. Le champ "identifiant" doit en quelque sorte être un résumé du champ "description". Nous développerons cet aspect "sémantique d'un nodule" dans la section 2.3.3.

a) Le champ "actions"

Nous avons distingué depuis le début de ce chapitre, quatre types d'actions. Le champ "actions" pourra donc contenir quatre types d'instructions :

- évaluer: ***nomProcédure*** **[sur:** ***nomObjet*****]**

- demander: ***nomEcran*** **[sur:** ***nomObjet*****]**

- déclencher: ***nomGraphe*** **[sur:** ***nomObjet*****]**

- déduire: ***nomInférence*** **[sur:** ***nomObjet*****]**

D'autre part, le modèle Situation-Action sur lequel nous nous appuyons n'interdit pas l'exécution de plusieurs actions depuis un nodule. Il est donc possible de définir plusieurs instructions à la suite dans le champ "actions" : les actions correspondantes sont toutes exécutées séquentiellement, puis le moteur choisit un nodule suivant.

b) le champ "règles de choix"

Le champ "règles de choix" contient un ensemble de règles indépendantes, affectées chacune d'un coefficient de priorité.

Une règle se présente de la manière suivante :

si: ***unePrémisse*** **alorsAller:** ***nomNodule*** **[priorité:** ***unChiffre*****]**

Si l'évaluation de *unePrémisse* [2] renvoie VRAI, le nodule *nomNodule* est associé à la priorité *unChiffre*. Après avoir parcouru toutes les instructions du champ "règles de choix", le moteur choisit le nodule le plus prioritaire. S'il y a plusieurs nodules au niveau de priorité le plus fort, le moteur en choisira un arbitrairement. La priorité la plus forte est 1. Ne pas spécifier de priorité à une règle équivaut à lui affecter la priorité la plus faible, c'est-à-dire 10.

La notion de priorité est importante dans notre approche de modélisation incrémentale parce qu'elle est un moyen d'exprimer (en principe provisoirement) une connaissance qui n'a pas encore été explicitée : la notion de priorité permet de préciser qu'une règle est prioritaire sur une autre en attendant d'identifier précisément le fait manquant qui permettra de lever l'ambiguïté.

Il est possible de définir également des règles inconditionnelles, car sans prémisse :

aller: ***nomNodule*** **[priorité:** ***unChiffre*****]**

2.2.1.3 Les actions

Par abus de langage nous appelons *action* non seulement l'exécution d'une procédure, mais aussi la procédure elle-même. Nous distinguons trois véritables types d'actions différentes : les procédures (action "évaluer"), les écrans (action "demander") et les inférences (action "déduire"). Les actions "déclencher" doivent être classées à part, parce qu'elles appellent des sous-graphes et ne font pas référence à des objets externes à la base de nodules.

Une action contient au moins un champ "identifiant" et un champ "description". Chaque action doit en effet posséder un identifiant unique : c'est un objet à part entière qui peut être consulté à l'intérieur d'une base de procédures, base d'écrans ou base d'inférences. Le champ "description", en plus du commentaire donné par le concepteur expliquant le rôle de l'action et décrivant explicitement ce qu'elle fait en langage naturel, doit contenir une description formelle des faits attendus en retour. Cette description est appelée *définition*.

a) Définition d'une action

Les faits attendus en retour de l'exécution de l'action doivent être déclaré de manière à connaître le nombre de faits qui peuvent être retournés et leur type.

---

[2] Exemples de formules :
(p1 ET f2) OU p3
NON(p1) ET p2 ET p3
où p1, p2 et p3 sont des propositions (par exemple : "conjoint.ouvrantDroit.age > 60"). Voir annexe 1.

Précisons qu'en plus de la valeur *nil*, quatre types sont définis dans EDINOS (booléen, chaîne, entier, réel) et qu'il est possible de préciser un domaine de valeurs pour un attribut, sous la forme d'une liste de valeurs possibles.

L'annexe 1 définit le langage de déclaration des faits.

b) Les écrans

Les écrans doivent contenir deux champs supplémentaires :

- un champ "texte" dans lequel le concepteur spécifie le message à afficher au minitéliste ou la question à lui poser.

- un champ "règles de traduction" qui définit comment traduire la réponse du minitéliste en fait renvoyé par l'action (ex: si le minitéliste tape 'O', 'o', 'OUI' ou 'oui', renvoyer le fait <confirmation>).

Pour clore cette présentation des spécifications de l'outil EDINOS, la figure 2.2 récapitule les différents objets qu'on trouve au sein de l'environnement EDINOS.

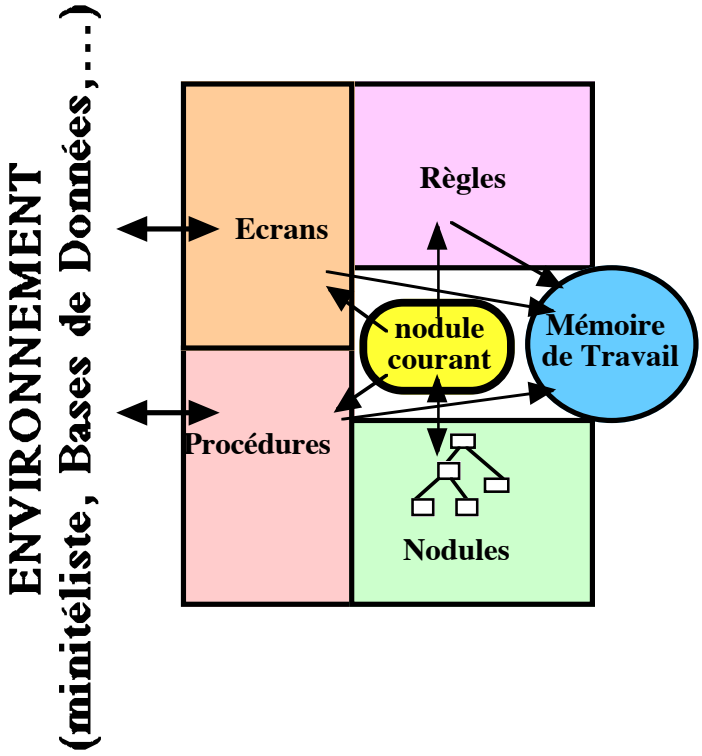


Figure 2.2 : Fonctionnement du moteur exploitant la BC

2.2.2 La réalisation finale

L'outil EDINOS a été développé à partir des spécifications données dans les sections précédentes en Smalltalk/V dans un environnement Mac/OS. Un certain nombre de fonctionnalités prévues dans les spécification n'ont pas été menées à terme :

- il n'a finalement pas été utile de développer une Base de règles au sein d'EDINOS : l'action de type "déduire" n'a pas eu de véritable application[3]. Ceci peut s'expliquer par le fait que la BC construite a finalement été de petite taille (voir annexe 3 pour les détails de la BC construite).

- Pour les mêmes raisons, dans le moteur exploitant la BC d'exploitation, les faits sont restés propositionnels. Si les notions d'objet, de relation, d'attribut ont été développées et expérimentées dans l'environnement EDINOS, elles n'ont pas été utilisées pour la BC d'exploitation.

Toutes les autres spécifications exposées dans les sections précédentes ont été réalisées dans EDINOS. En outre, les fonctionnalités suivantes ont été développées :

a) Aides à la création d'objets

Pour accélérer la "programmation" des nodules de situation, l'interface d'EDINOS offre au concepteur un véritable "environnement de construction de graphes de nodules". Dans la fenêtre affichant le contenu d'un nodule (la figure 2.4 montrera une fenêtre de ce type), le concepteur a sous la main un ensemble de boutons correspondants aux jeux d'instruction des champs "actions" et "règles de choix" (exemple : "demander:", "alorsAller:",...). La liste des objets qui peuvent servir à la programmation est affichée dans la fenêtre. Par exemple, si le concepteur clique sur le bouton "demander:", l'instruction s'inscrit automatiquement dans le champ "actions" et la liste des écrans déjà définis apparaît : le concepteur n'a plus qu'à cliquer sur un nom d'écran pour remplir le champ "actions". Ce type de fonctionnalité, qui permet de créer et de remplir des nodules de situations en quelques "clics" de souris, n'est pas anecdotique car si notre approche incrémentale exige que la révision de la BC soit aisée, cela implique que la création

[3] Plus exactement : il y aurait eu une application possible (donner l'adresse d'un centre d'examen de santé à partir du département d'habitation de l'affilié), mais il aurait pour cela fallu développer spécifiquement pour cet exemple les deux moteurs déductifs nécessaires (au sein d'EDINOS et au sein du serveur télématique), ce qui n'a pas été jugé prioritaire...

(l'enrichissement fait partie du processus de révision) des objets de la BC doit également être rendue aussi facile que possible.

b) La notion de "procédure attachée" à un écran

Tel que nous l'avons défini dans la section 2.1.3.1, un écran est un objet au contenu statique. Pour permettre d'afficher à l'intérieur d'un écran des champs dynamiques (par exemple une date), nous avons introduit la notion de "procédure attachée". Une procédure attachée, de même qu'une procédure normale, est définie conjointement par le concepteur et l'informaticien capable de développer des programmes sur le serveur télématique. L'exécution d'une procédure attachée est effectuée au cours de l'exécution de l'action "demander" et provoque l'affichage dynamique d'un texte à l'intérieur de l'écran (au contenu statique), à l'endroit précis indiqué par le concepteur au moyen d'un symbole spécifique.

2.2.3 Un exemple

Afin d'illustrer les fonctions et objets présentés jusqu'à présent, de mieux comprendre le système en fonctionnement et d'avoir un aperçu de l'interface d'EDINOS, nous allons nous arrêter dans cette section sur un exemple.

a) Un graphe

Le graphe de la figure 2.3 est le graphe "correspondanceCP" destiné à donner au minitéliste les coordonnées du bureau compétent de la Caisse de Prévoyance pour la question qu'il veut régler. Par un dialogue qui consiste à poser au plus quatre questions au minitéliste, le système cherche à cerner le type de problème qui concerne le minitéliste pour l'orienter sur le bureau administratif adéquat. A la fin de ce dialogue, le système affiche les coordonnées postales et téléphoniques du bureau et propose au minitéliste de poser sa question via le système de messagerie.

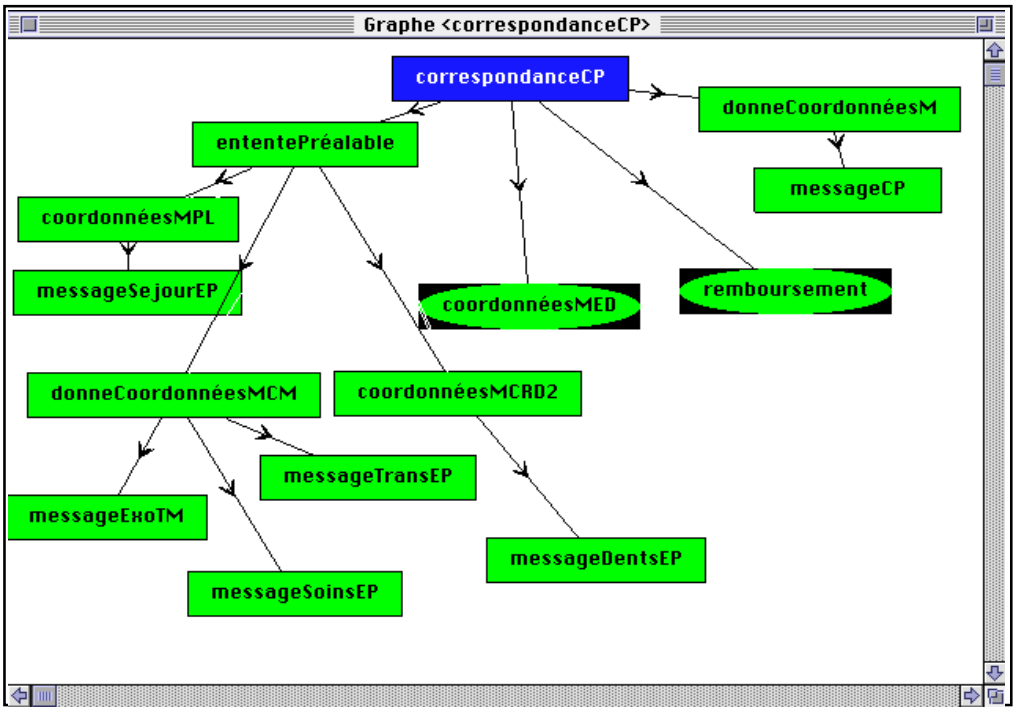


Figure 2.3 : Le graphe “correspondanceCP”

Le nodule initial “correspondanceCP” apparaît dans la figure 2.4. L’écran “typeProblèmeCP” appelé par le nodule “correspondanceCP” est affiché dans la figure 2.5.

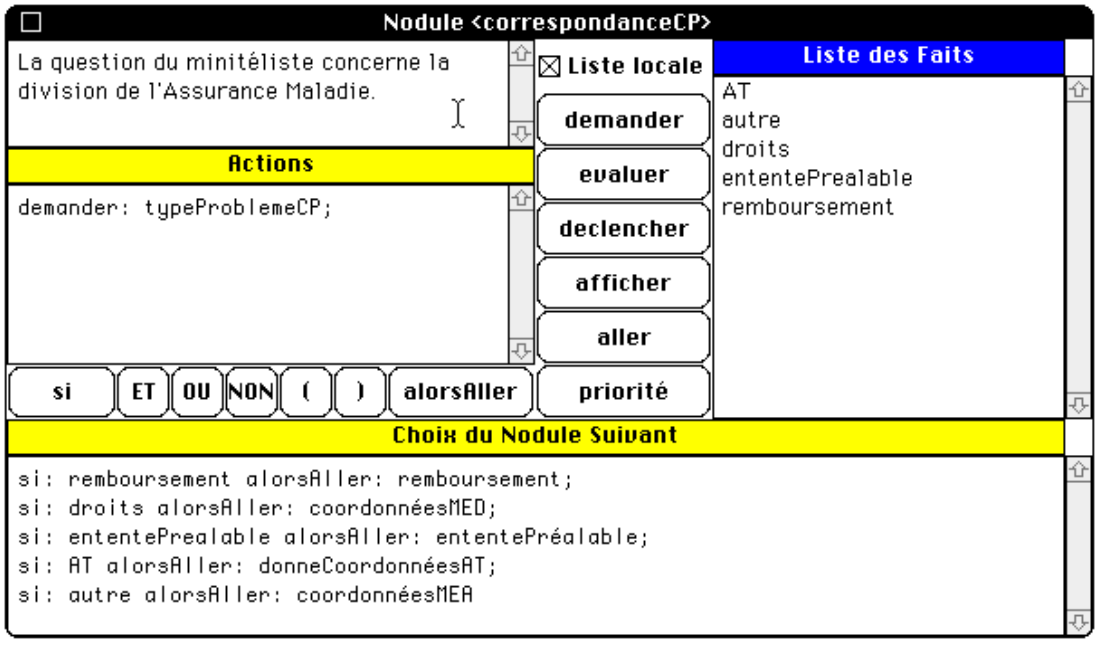


Figure 2.4 : Le nodule “correspondanceCP”

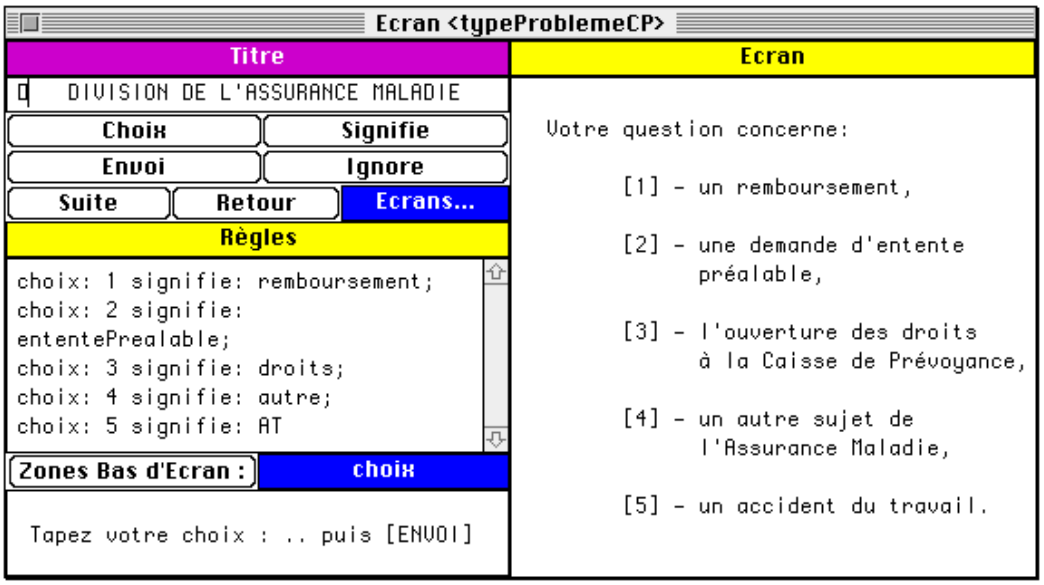


Figure 2.5 : L'écran “typeProblèmeCP”

b) Interface de l’outil de simulation du dialogue

La figure 2.6 correspond à la simulation de l’écran d’un nodule terminal, affichant les coordonnées d’un bureau.

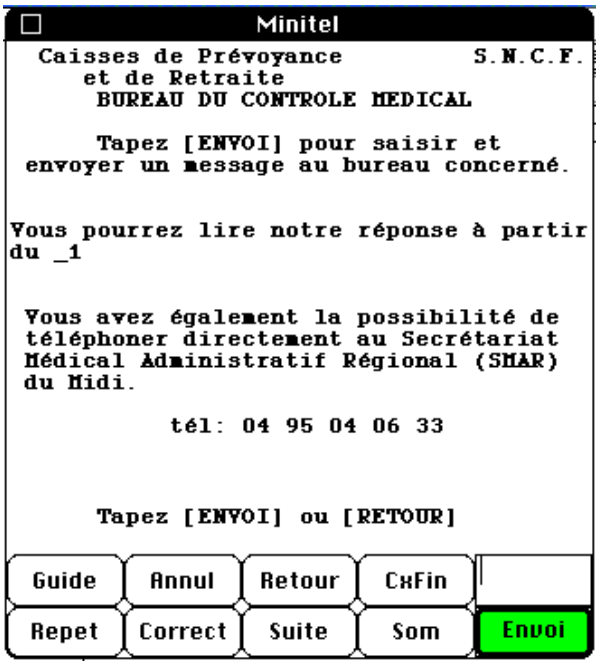


Figure 2.6 : Simulation d’un écran

La fenêtre de la figure 2.7 simule l'exécution de la procédure <numeroGT> qui calcule le numéro du bureau GT à partir du dernier chiffre du numéro d'immatriculation de l'affilié.

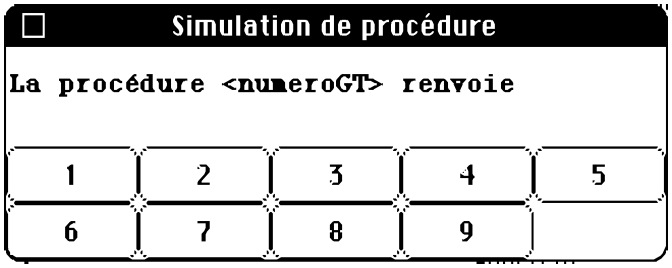


Figure 2.7 : Simulation de la procédure "numeroGT"

Au cours de la simulation, le concepteur peut interrompre le moteur pour examiner la situation courante. Une fenêtre du type de celle de la figure 2.8 s'ouvre alors :

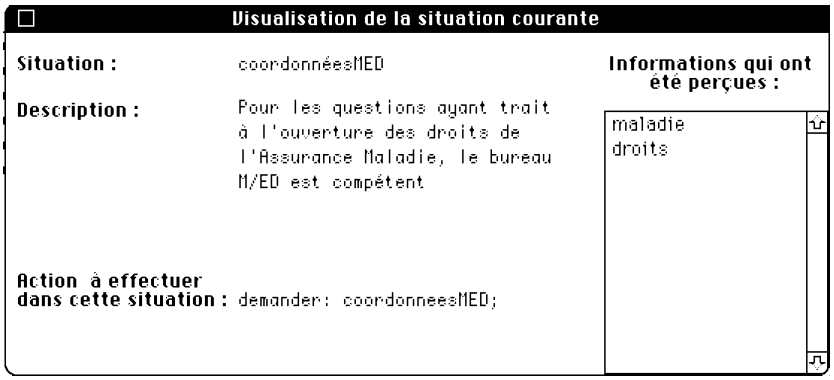


Figure 2.8 : Fenêtre "Visualisation de la situation courante"

2.3 Une méthodologie

Il est nécessaire d'accompagner l'outil EDINOS, tel qu'il a été défini dans la section 2.2, et son guide-utilisateur, d'un "mode d'emploi des nodules de situation" afin d'aider et guider le concepteur dans la construction de graphes de nodules. Ce mode d'emploi a été progressivement construit à partir des leçons concrètes tirées de l'expérience acquise dans la construction de BC sous forme de nodules de situation. L'ensemble de ces leçons esquisse une *méthodologie* pour la conception incrémentale de BC sous forme de nodules de situations.

La première partie de ce mode d'emploi a pour but de donner au concepteur / utilisateur d'EDINOS, une vision claire de ce qu'est une situation et un nodule de situation.

2.3.1 Comprendre la situation ?

a) Retour sur la notion de situation

Proposer de s'appuyer sur la notion de situation pour représenter des savoir-faire requiert d'avoir une vision claire de cette notion. A cet effet, nous proposons de distinguer quatre types de situations :

- la **situation réelle**, qu'il est préférable d'appeler simplement **état du monde** pour rester cohérent avec la définition donnée dans la section 1.3.3 (la situation comme interprétation de l'environnement au cours de l'accomplissement d'une tâche[4]). Exemple : "Monsieur Dupond, assis devant son Minitel dans son salon, veut savoir s'il a droit à un bilan de santé" est une description d'une situation réelle, qui en l'occurrence est fictive. Le propre d'une situation réelle est qu'on peut n'en faire que des descriptions (des modélisations).

- la **situation concrète** est une représentation, que *se construit le concepteur* ou cogniticien, d'une ou plusieurs situations réelles. Exemple : "cas de Monsieur Dupond qui veut savoir s'il a droit à un bilan de santé". Rappelons que le projet ou la tâche en cours du concepteur est une composante de la situation concrète ;

- la **situation courante** est une représentation *dans le langage de la BC* d'une situation concrète. Elle est appelée ainsi parce qu'elle n'existe que fugitivement *au cours* d'une exécution du moteur. Exemple : "cas d'un retraité qui veut savoir s'il a droit à un bilan de santé" ;

- le **nodule de situation** est une représentation dans la BC d'un ensemble de situations courantes possibles donc d'un ensemble de situations concrètes. Un nodule de situation est donc une abstraction de situation concrète. Il est important d'insister sur le fait qu'un seul nodule peut représenter plusieurs situations concrètes. Exemple : on peut créer un nodule "cas d'un minitéliste qui veut savoir s'il a droit à un bilan de santé" sans jamais avoir besoin de faire une distinction conceptuelle au sein du système entre les deux situations concrètes "cas d'un retraité qui veut savoir s'il a droit à un bilan de santé" et "cas d'un agent en activité qui veut savoir s'il a droit à un bilan de santé"...

---

[4] Comme cette notion de "situation réelle" correspond à la vision courante de la notion de situation, il n'est pas scandaleux d'utiliser ce terme, qui pourtant est une aberration si l'on suit notre définition.

On voit dans cet exemple que le formalisme de représentation par nodules de situation permet de représenter un même savoir-faire sous la forme d'une multitude de BCs différentes. En effet, il est tout à fait possible de chercher à créer un nodule de situation pour chaque situation concrète, que celle-ci soit imaginaire ou fortement ancrée dans la réalité. Il est donc nécessaire au concepteur de faire des choix de conception lors de la construction de la BC. Pour aider à faire ces choix, nous proposons une méthodologie de construction de BC sous forme de nodules de situation. Cette méthodologie a été progressivement formulée à partir de l'expérience de construction de graphes de nodules de situation.

b) Comprendre la situation courante

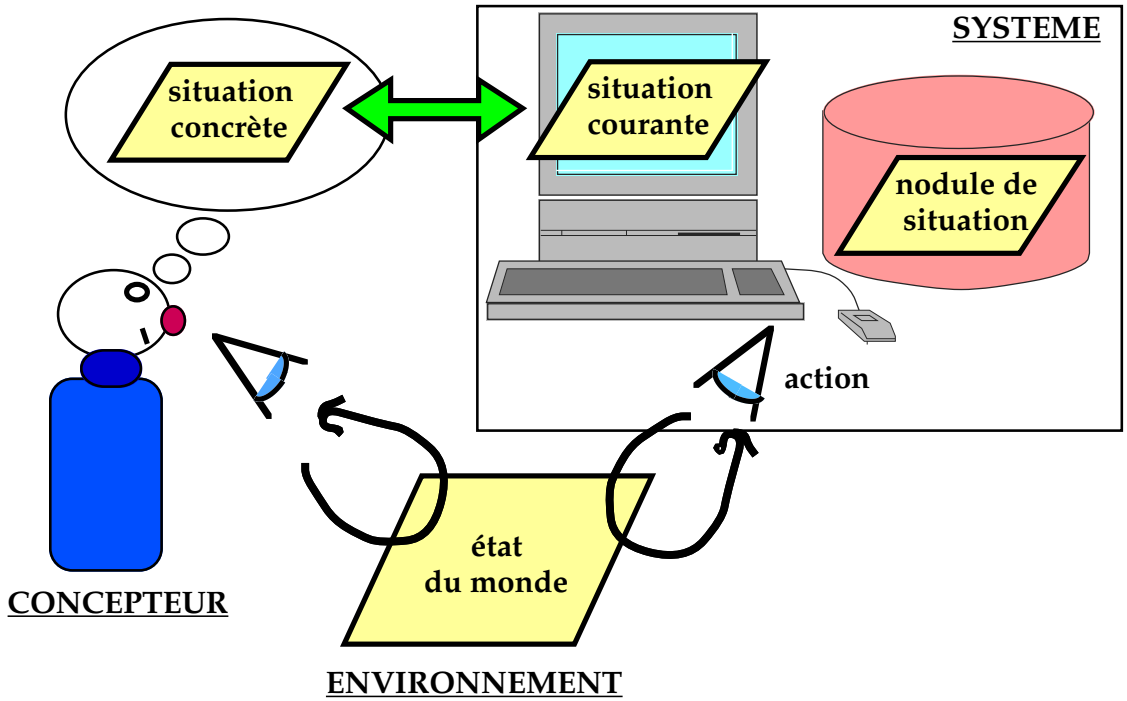


Figure 2.9 : Les quatre types de situation

Les quatre types de situations sont représentés dans la figure 2.9. C'est la trilogie situation concrète - situation courante - nodule de situation qui nous intéresse particulièrement. En effet, la construction de la BC est effectuée *via* un concepteur, filtre incontournable entre le système et la réalité. Le concepteur a une situation en tête (situation concrète) qu'il voudrait voir représentée dans le SBC sous

la forme d'une situation courante. C'est cette situation concrète qui est centrale dans le processus. Il est même possible de se passer de la situation réelle (état du monde) : l'outil de simulation permet au concepteur de donner lui-même au système les caractéristiques de la situation réelle. La situation concrète se substitue alors complètement à la situation réelle dans le processus de modélisation. Par la suite nous ne ferons plus référence à la situation réelle, mais uniquement à la situation concrète.

Si l'on cherche à caractériser d'une manière générale le problème de la compréhension du système par le concepteur, on peut proposer qu'il s'agit à la base d'un problème de mise en correspondance entre situation concrète et situation courante. Lorsque le moteur est lancé et donne un résultat (c'est-à-dire propose une action) pour la situation courante, cette situation sera comprise par le concepteur s'il parvient à faire la correspondance avec une situation concrète qu'il construit mentalement. Cette correspondance est plus facile à faire lors d'une simulation, car le concepteur se construit la situation concrète progressivement au fur et à mesure de la simulation. A l'opposé, afficher directement au concepteur un écran décrivant la situation courante demande au concepteur de reconstruire instantanément une situation concrète (éventuellement en effectuant mentalement le même type de simulation que le système).

Une situation courante peut être définie formellement par la conjonction de trois éléments :

- un nodule courant,
- l'état de la MT,
- une "trace", que nous définissons comme la liste ordonnée des nodules précédemment activés. La trace du graphe courant comprend (éventuellement récursivement) la trace du nodule appelant le graphe.

Dans l'interface EDINOS, la situation courante est présentée dans une fenêtre (voir figure 2.8) affichant le contenu de la MT, l'action et la description de la situation (le champ "commentaires" du nodule). Les fonctionnalités de "compréhension par exploration" permettent à l'utilisateur d'accéder aux actions à l'origine de chaque fait.

Il nous faut bien préciser que l'objet situation courante correspond à une interruption du moteur **au cours** de l'exécution de l'action du nodule courant. Le nodule suivant ne peut, par conséquent, pas être connu puisque l'action courante n'a pas encore renvoyé de fait. Du point de vue de l'interface du système, l'utilisateur accède à la fenêtre de la figure 2.8 en interrompant l'exécution d'une action (le système lui posait une question correspondant à l'action du nodule courant).

Notre objet “situation courante” correspond au modèle du cas au sens de Walter Van de Velde (voir section 1.2.1.3) : il reflète l’ensemble des connaissances sur le problème et sa résolution, pas seulement les données sur le cas. Le nodule de situation dans ce contexte peut être vu comme une abstraction de modèle du cas : une sorte de “modèle de modèle du cas” si l’on peut dire, autrement dit un modèle du cas réutilisable.

Joost Breuker décompose la solution complète d’un problème en trois parties qui rejoignent beaucoup les composants de notre situation courante.

“Nous proposons qu'une solution complète ait trois composantes. (1) Le modèle du cas représente la compréhension d'un problème. (2) La conclusion est la réponse à la question posée par la définition du problème, tandis que (3) la structure d'argument établit la démonstration qui soutient la conclusion. La conclusion fait à la fois partie de la structure d'argument et du modèle du cas.”

[Breuker, 1994, p. 123]

La figure 2.10, tirée de [Breuker, 1994], illustre cette décomposition.

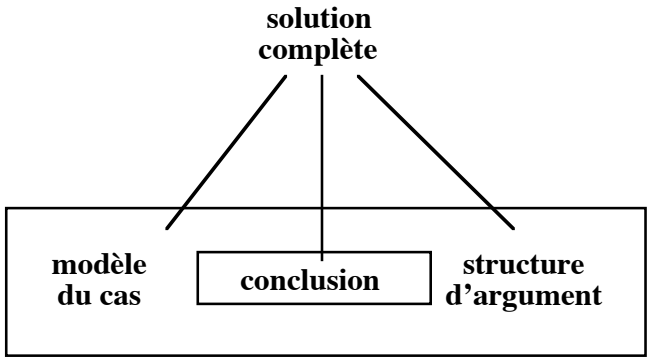


Figure 2.10 : Les composants d’une solution ([Breuker, 1994])

“Les rôles des trois composants d'une solution, en ce qui concerne la communication avec l'utilisateur, sont les suivants :

- Explication : le MODÈLE DU CAS doit expliquer les données.

- Justification : la STRUCTURE D'ARGUMENT doit justifier la conclusion.

- Résultat : la CONCLUSION est un point d'entrée pour une autre définition de problème, ou (sous)tâche. Cette (sous)tâche peut être exécutée par un autre agent que celui qui a produit la solution”

[Breuker, 1994, p. 124]

Il est possible de faire l’appariement suivant avec les composants de notre situation courante :

- la conclusion correspond à l'action[5] du nodule courant,
- la structure d'argument correspond à la trace,
- le modèle du cas correspond à l'état de la MT et au nodule courant.

La compréhension de la situation courante est une action de modélisation effectuée par le concepteur : c'est bien lui qui va établir une correspondance avec une situation qu'il se construit en adaptant le niveau d'abstraction de cette situation concrète qu'il a en tête. Cette flexibilité conceptuelle du concepteur lui permet de se placer au bon niveau d'abstraction afin de permettre une correspondance un à un entre la situation courante et la situation concrète. Dans l'exemple donné ci-dessus, la situation concrète était : "cas de M. Durand", la situation courante : "cas d'un retraité". Le concepteur ne jugeait pas utile de différencier les différents cas de retraités au sein de la BC[6]. Lorsqu'il (ou un autre) est ensuite amené à comprendre la situation courante "cas d'un retraité" au cours de l'exécution du moteur pour M. Durand, il est confronté à une situation courante qui est définie par exemple par les informations suivantes : "statut : retraité, région d'habitation : centre, age : 60" où ne figurent pas le nom du retraité puisque cette information est inutile... Il doit donc abstraire sa représentation de situation concrète de "cas de M. Durand" à "cas d'un retraité". Si le concepteur ne parvient pas à faire cette correspondance unique, triviale dans notre exemple, c'est-à-dire s'il ne parvient pas à construire une situation concrète équivalente à la situation courante, il peut s'agir d'un symptôme de BC incorrecte. Nous reviendrons sur cet aspect dans le chapitre 3. Pour l'heure, nous considérerons que le concepteur a la charge de :

- créer un langage adéquat de description des situations courantes (via la définition des actions et des faits retournés), pour que toutes les situations concrètes significatives puissent être représentées au sein du SBC;
- se construire des situations concrètes au niveau d'abstraction des situations courantes pour être capable de comprendre le système en fonctionnement.

Si ces conditions sont respectées, et si la BC a été "bien conçue", on peut considérer que situation courante et situation concrète sont équivalentes (la première est une représentation de la seconde), du point de vue de la tâche de conception.

---

[5] On peut noter que la notion de conclusion présentée par J. Breuker fait écho au type d'action "déclencher" qui permet d'appeler une sous-tâche. Si on ne cherche pas à coller cette structure de solution complète à notre objet "situation courante" (autrement dit si on prend la "photo" du processus à un autre moment), on pourrait aussi faire un lien entre cette notion de conclusion et le nodule successeur choisi.

[6] Le concepteur aurait tout à fait pu faire un autre choix et distinguer les deux cas de retraités "cas de M. Dupond" et "cas de M. Durand" par exemple, et s'attacher à ce qu'il puisse y avoir deux situations courantes différentes pour les deux retraités...

c) Comprendre un nodule de situation

Une situation courante est une instanciation d'un nodule de situation au cours d'une exécution. Cette instanciation se fait en complétant le nodule courant par :

- un état de la MT,
- une trace dans le réseau de situations.

Un nodule de situation est donc plus abstrait, moins "situé" qu'une situation courante, au sens où les éléments concrets de l'environnement (éléments de la MT) ne font pas partie du nodule de situation. On peut par conséquent voir le nodule de situation comme la partie abstraite (réutilisable) de la situation courante...

Il n'est donc en général pas possible de faire une correspondance un à un entre un nodule de situation et une situation courante. Pour cette raison, comprendre un nodule de situation n'est pas tâche aisée pour le concepteur, d'autant plus si ce dernier n'est pas le créateur du nodule. La tâche d'abstraction de la situation concrète effectuée par le concepteur pour comprendre une situation courante doit être poursuivie encore plus loin s'il veut comprendre le nodule de situation[7].

Mais un nodule de situation peut aussi être vu comme un agrégat de plusieurs situations courantes différentes. Donc si la tâche de compréhension directe d'un nodule est difficile, elle peut être facilitée par les compréhensions individuelles de toutes les situations courantes qui correspondent au nodule... Nous développerons cet aspect dans la section 3.3.2.2 où nous introduirons la notion *d'exemples abstraits* qui sont des objets d'un niveau d'abstraction intermédiaire entre situation courante et nodule de situation. Un exemple abstrait est un ensemble de situations courantes possibles, correspondant à la même trace dans le réseau de nodules. Pour éviter au concepteur d'avoir à comprendre le nodule courant, voire l'ensemble du réseau de nodules, la révision coopérative va chercher à s'appuyer sur ces exemples abstraits pour dialoguer avec le concepteur et chercher à corriger automatiquement le graphe courant.

Une fois effectué cet éclaircissement préalable sur la notion de situation, nous sommes maintenant en mesure d'exposer étape par étape la méthodologie de construction incrémentale de la BC.

2.3.2 Démarche générale de construction de la BC

[7] On pourrait introduire la notion de "situation abstraite" pour définir la représentation que se construit le concepteur pour comprendre le nodule de situation. Comme son nom l'indique, une situation abstraite serait une abstraction de situation concrète.

Conformément à la démarche définie dans le chapitre 1, la méthodologie de construction des graphes de nodules repose sur l'incrémentalité : le concepteur se penche successivement sur différents “cas résolus” qu’il représente dans la BC.

a) Le nodule initial

Le concepteur construit un graphe pour une finalité précise : le graphe correspond à une tâche qui répond à un but ou une attente. La première étape consiste donc à créer un nodule initial, expression de ce but. C'est à travers le nom du nodule et le contenu de son champ “commentaires” que sont décrits explicitement le but et le cadre d'application du graphe.

Exemple : création d'un nodule initial “allerAuCinéma” avec dans le champ commentaires “Le but est d'aller assister à une séance de cinéma. Au départ, je suis à la maison”

b) L'incrémentalité méthodique

L'approche incrémentale nécessite de procéder rigoureusement au cas par cas : situation après situation. Il est donc recommandé au concepteur de procéder de manière récursive selon la méthode décrite dans la figure 2.11 :

étape 1 : choisir une et une seule situation concrète.
étape 2 : représenter la situation concrète dans la BC, ainsi que l'enchaînement des situations qui mène à cette situation.
étape 3 : recommencer à l'étape 1 avec une autre situation concrète.

Figure 2.11 : L'incrémentalité méthodique

La situation concrète choisie lors de l'étape 1 correspond généralement à un cas résolu. En effet, le concepteur construit la BC pour donner des solutions complètes, pour exécuter une tâche de bout en bout : la situation concrète qu'il veut représenter est une expérience de réalisation de la tâche.

L'étape 1 ne pose en principe pas de problème pour la première situation concrète, au tout début du processus. En général, le concepteur qui a une tâche en tête et qui veut la représenter, connaît également au moins une résolution concrète de cette tâche. Mais cette étape 1 n'est pas toujours triviale et nous reviendrons sur ce

point plus tard, lorsqu'il s'agira de construire des situations concrètes alternatives à celles déjà représentées.

L'étape 2 est le cœur de la construction de la BC : nous allons la détailler au point suivant.

Lorsqu'au cours de l'étape 2, une situation voisine vient à l'esprit du concepteur, celle-ci doit être notée à part et explicitement décrite, afin d'être traitée plus tard. Cette nouvelle situation ne doit pas interrompre le processus de modélisation complète de la situation concrète en cours. Il est important de traiter complètement chaque situation concrète avant de passer aux autres, sinon le risque est grand de mélanger les différentes situations simultanément envisagées. L'avantage de s'en tenir à une seule situation est de se raccrocher au côté concret de cet objet à modéliser : le concepteur part probablement d'une situation réelle pour laquelle il veut que le système fonctionne correctement. C'est cet ancrage dans la réalité qui guide et valide le processus de modélisation et d'abstraction de situations.

Tout au long de cette section, l'exemple ci-dessous va nous servir d'illustration :

Exemple de situation concrète : “La dernière fois que je suis allé au cinéma (j'avais choisi de voir “Capitaine Conan” ; il passait à Brétigny ; j'y suis allé en voiture)”.

---

c) Modéliser la première situation concrète (étape 2)

c1) Bâtir le squelette du graphe

Au départ, seul le nodule initial a été créé. Le concepteur se représente une situation concrète, un cas résolu, qu'il voudrait voir représenté dans la BC. Il va donc falloir créer les nodules et les actions d'un squelette de graphe, qui mène du nodule initial à un nodule terminal correspondant à la situation concrète.

L'ordre des actions à effectuer pour parvenir à la situation voulue va guider naturellement la construction du squelette initial du graphe. Le concepteur doit se poser les questions suivantes :

- quelles sont les actions qu'il faut nécessairement effectuer entre la situation initiale et la situation finale ?
- dans quel ordre ?

En général, ce découpage en actions successives est un découpage de la tâche en étapes successives, qui chacune correspond à un sous-but. Dans un premier temps, il est conseillé de définir les actions représentant ce sous-but comme des procédures (compétences compilées, donc non explicites). On peut se contenter

à cette étape de simplement créer les procédures, sans encore décrire les faits renvoyés. Par contre, il est recommandé d'expliciter les différentes situations correspondant aux différents nodules : décrire dans le champ “commentaires” de chaque nodule la situation représentée et les choix de conception éventuels qui ont été faits (où en est-on dans la tâche ? A quelle étape correspond le nodule ?). Il convient de trouver un nom explicite qui résume au mieux la description du nodule.

Exemple de nodule : nom = “choisirMoyenTransport”, commentaires = “La séance a été trouvée, il faut maintenant chercher un moyen de transport”.

A la fin de cette étape, le concepteur a bâti un squelette initial, basé sur notre situation concrète : un graphe qui est un simple enchaînement de nodules menant du nodule initial à un nodule final. A chaque nodule, il a associé une action. Pour l'instant les règles de choix n'ont pas été précisées et le graphe n'est pas encore opérationnel.

La situation concrète “la dernière fois que je suis allé au cinéma” est un cas résolu (le moment durant lequel j'étais assis dans la salle de projection), et je suis capable de donner l'ordre des actions (ou des problèmes) significatives qui se sont déroulées avant d'arriver à cette situation (choisir le film ; choisir la salle ; choisir le moyen de transport ; aller à la salle ; acheter le ticket ; s'installer dans la salle). On voit dans cet exemple[8] que la modélisation nécessite de bien cerner le domaine cible, de savoir précisément au départ ce qu'on a besoin de représenter, jusqu'à quel niveau de détail aller, etc.

c2) Remplir le squelette

Il faut maintenant remplir le moule qui a été défini précédemment. Le but de cette étape est d'obtenir un graphe opérationnel et correct pour la situation concrète courante. En balayant le graphe à partir du nodule initial, le concepteur doit pour chaque nodule :

1) définir complètement l'action ou les actions,

2) définir la règle de choix pour la situation concrète.

A ce stade, l'action attachée au nodule a déjà normalement été créée. C'est maintenant le moment de définir précisément les faits que peut alternativement renvoyer l'action, dans son champ “description”.

[8] Les exemples comme celui-ci tirés de la vie de tous les jours montrent vite leurs limites : il est peut-être impossible de trouver dans notre exemple des actions qu'on peut considérer comme des procédures informatiques réalistes.

Ensuite, le champ “Règles de choix” peut être complété en créant la règle qui, pour l'ensemble des faits correspondants à la situation courante (prémisse de la règle) permet d'activer le nodule suivant du graphe.

A cette étape, lorsque le concepteur visualise les autres faits possibles, il est tenté d'envisager des alternatives. Par exemple, “que faire si aucune salle ne passe le film choisi ?”. Ces alternatives sont donc des pistes pour développer d'autres branches du graphe. Ce ne sont pas encore précisément des cas résolus : nous reviendrons sur ce point au paragraphe d). Il est conseillé de terminer cette étape de remplissage du squelette avant de passer à une nouvelle situation concrète.

Exemple : pour le nodule “choixDeLaSalle”, l'action “choisirLaSalle” renvoie le fait "salleChoisie” ou nil. La règle “si: salleChoisie alorsAller: choixMoyenTransport” est créée par le concepteur.

---

c3) Développer un sous-graphe

En fin d'étape c2, la situation concrète initiale est représentée plus ou moins complètement. En effet, les sous-tâches générales qui ont pu être dégagées ont été plus ou moins développées. Certaines tâches ont été provisoirement exprimées sous la forme d'une procédure. Plutôt que de passer à une nouvelle situation concrète correspondant à un nouveau cas résolu, le concepteur peut faire le choix d'un développement qu'on peut qualifier d'en “profondeur d'abord” : expliciter chacune des actions du graphe, en créant éventuellement des sous-graphes.

Inversement, le concepteur peut préférer développer d'abord complètement le graphe courant, c'est-à-dire s'en tenir dans un premier temps au graphe principal et différer l'explicitation des actions, donc des sous-graphes éventuels. Il choisit alors une situation concrète parmi les alternatives qu'il a relevées. La place de cette étape c3 est donc laissée à l'initiative du concepteur, qui peut choisir de procéder plutôt graphe après graphe ou vraiment situation après situation. Ceci n'est pas en contradiction avec le schéma général d'incrémentalité méthodique de la figure 2.11. Simplement le concepteur est libre de fixer jusqu'à quel niveau de profondeur il veut pousser l'étape 2 (représenter le chemin qui mène au cas résolu). Le développement “en largeur d'abord” (s'en tenir à un graphe donné avant d'expliciter en détail les situations) se défend dans la mesure où il permet au concepteur de se concentrer sur un même niveau d'abstraction (un graphe, une tâche) et de le détailler, avant de passer à un autre niveau. La contrepartie de ce procédé est qu'il faudra envisager plusieurs fois la même situation concrète : chaque développement de sous-graphe

devra être validé par les situations concrètes qui ont servi à la construction du graphe principal.

c4) Tester la situation concrète

Qu'on ait ou pas effectué l'étape c3, on dispose dans tous les cas en fin d'étape c2 d'un graphe opérationnel qui doit correspondre à la situation concrète envisagée au départ (cas résolu). L'outil de simulation d'EDINOS permet de vérifier cela (cf. section 2.2.1.1). Le concepteur vérifie que la situation courante en fin d'exécution correspond à la situation concrète imaginée et que l'action proposée par le système est la bonne.

d) Choisir une autre situation concrète (étape 3)

Cette tâche n'est pas toujours aisée si le concepteur ne maîtrise pas parfaitement le domaine car il s'agit d'une tâche qui peut mettre en jeu les compétences du domaine. Le concepteur part d'un nouveau cas de figure, une alternative choisie, qu'il doit transformer en cas résolu, sinon la représentation de l'alternative au sein de la BC ne présente pas beaucoup d'intérêt[9]. Il faut donc que le concepteur à partir d'une éventualité (que se passe-t-il si...?), d'un problème, parvienne à identifier une solution précise, correspondant à une particularisation de l'alternative. Une alternative donne lieu en effet généralement à plusieurs cas résolus différents.

Lorsque le concepteur est capable de construire des situations concrètes à partir des alternatives, il a le libre choix de l'ordre des alternatives à envisager. Il est alors conseillé de commencer par les situations conceptuellement proches de la situation concrète initiale, c'est-à-dire correspondant aux alternatives du nodule final, puis à celles de son père, et ainsi de suite en remontant le graphe.

Sinon, lorsqu'il n'est pas possible au concepteur de construire des situations concrètes “à la demande”, celui-ci doit se contenter d'envisager les situations concrètes au fur et à mesure qu'elles se présentent à lui. C'est le cas lorsque le concepteur n'est pas expert du domaine. Son rôle est alors de détecter des alternatives et de les présenter à l'expert, qui lui donnera en retour les cas résolus correspondants.

---

[9] Se contenter de créer un nodule (vide) pour représenter l'alternative ne mène pas très loin, mais c'est équivalent à se contenter de créer un nodule initial pour un graphe sans aller plus loin. Cela peut être une manière *d'annoter* la BC qui signifie : “je connais l'existence de cette alternative, j'y reviendrai plus tard”.

Dans notre exemple, prenons l'alternative “la salle n'a pas pu être choisie”. C'est une situation. Mais ça n'est pour l'instant pas un cas résolu. Que doit-on faire dans cette situation ? En tant qu'expert du domaine, je suis capable de spécifier que si le film ne passe pas dans une salle qui me convienne (salle trop loin de chez moi, horaires inadéquats, ...), alors je dois renoncer[10]. L'alternative “la salle n'a pas pu être choisie” ne donne lieu qu'à une seule situation concrète : “renoncer à aller au cinéma”.

e) Modéliser une autre situation concrète

Lorsqu'une nouvelle situation concrète a été sélectionnée, on retourne à l'étape 2. Mais la différence essentielle avec la modélisation de la première situation concrète (c), est qu'on dispose désormais d'un graphe : le sous-graphe correspondant à cette nouvelle situation concrète va être attaché quelque part dans ce graphe.

Pour détecter le nodule à partir duquel il va falloir attacher le nouveau sous-graphe, le concepteur peut utiliser l'outil de simulation, appliqué à la nouvelle situation concrète. Le principe est de détecter le moment à partir duquel le graphe est incorrect pour la situation concrète courante. Le concepteur se laisse guider par le graphe, en simulant les différentes actions des nodules activés et interrompt le système dès lors qu'il n'est plus d'accord sur l'action à exécuter. A cet instant précis, le nodule courant n'est pas adéquat : il ne correspond pas à la situation concrète. Par contre, son père était encore “pertinent”. C'est lui qui va servir de point de départ au sous-graphe qui va être construit.

On est ramené donc au même schéma qu'au c), à la différence près que le squelette qu'il faut bâtir, mène d'un nodule bien identifié du graphe à un nodule terminal correspondant à la situation concrète. Les étapes identifiées au c) peuvent se succéder de la même manière :

- c1) Bâtir le squelette,
- c2) Remplir le squelette,
- c3) Développer éventuellement les sous-graphes,
- c4) Tester la situation concrète.

Le processus de simulation pour identifier le point d'accrochage du nouveau sous-graphe peut être effectué virtuellement : le concepteur peut parcourir

[10] Bien entendu, nous faisons ici une simplification extrême qui ferait pleurer n'importe quel cinéphile, car il faudrait plutôt spécifier que dans cette situation il convient de chercher un autre film... Nous serions alors confronté à un problème d'itération, puisqu'il faudrait revenir à l'étape de choix du film, en gardant en mémoire que le premier film choisi ne convient pas, et mettre en oeuvre une méthode type genérer-et-tester pour le choix du film. Nous évoquerons plus tard le problème de l'itération, extension future des graphes de nodules.

mentalement le graphe... L'outil de simulation informatique fournit un moyen de contrôler exhaustivement que la situation concrète courante colle complètement à la partie “commune” du graphe.

Ceci conclut la présentation de la méthodologie de construction incrémentale des graphes de nodules. En parallèle à l'expérimentation de cette méthodologie, nous avons élaboré et adopté des conventions pour la dénomination des nodules. Ces conventions ont pour objectif de rendre plus aisé la compréhension de la BC, donc de faciliter aussi sa construction.

2.3.3 Conventions pour la dénomination des nodules

La signification de chaque nodule dépend de sa position dans son graphe (quels nodules ancêtres ? quels nodules descendants ?), ainsi que de son contenu (actions et règles de choix).

Trois dimensions sont donc à distinguer dans la sémantique d'un nodule :

1) une dimension BILAN, relative aux nodules ancêtres et au chemin parcouru depuis le nodule initial ;

2) une dimension ACTIVITÉ : le contenu du nodule, ce qu'il fait à proprement parler (actions et règles de choix) ;

3) une dimension PROJET : quelle est la nouvelle formulation du projet courant. Le projet courant a pu en effet se modifier, se préciser depuis le projet, sûrement plus général, exprimé dans le nodule initial. Il sera réalisé par l'ensemble des nodules descendants.

La sémantique de chaque nodule de situation dépend de ces trois composantes. Un nodule initial n'a pas de dimension “bilan”, un nodule terminal n'a pas de dimension “projet”.

Le nom du nodule ne peut exprimer qu'une seule de ces dimensions, et le concepteur doit faire un choix, de manière à retenir celle qu'il pense être la plus significative (voir figure 2.12 : un même nodule peut être nommé au moins de trois manières différentes).

NOM du NODULE	DIMENSION	EXEMPLE
verbe au participe passé, ou adjectif qualificatif	BILAN	“parti avant 55 ans”
verbe au présent	ACTIVITÉ	“vérifie grade”
verbe à l'infinitif	PROJET	“chercher droits à pension”

Figure 2.12 : Une convention pour la dénomination des nodules

On retrouve ce problème de la dénomination des objets dans d'autres méthodologies d'acquisition des connaissances. Ainsi, [Cañamero, 1995] remarque que les inférences sont décrites dans KADS avec un point de vue opératoire (l'équivalent de notre dimension ACTIVITÉ), tandis qu'elles sont décrites dans CommonKADS selon un point de vue téléologique (en fonction du but qu'elles permettent d'atteindre, ce qui correspond à notre dimension PROJET).

L'application des conventions mentionnées ci-dessus conduit naturellement à proposer des noms contenant des verbes à l'infinitif pour les nodules initiaux (et donc pour les graphes). Si un nodule doit appeler un graphe, il peut être logique de le nommer par un verbe conjugué au présent de l'indicatif, pour signifier que l'action est *en cours* d'exécution. Par exemple : le nodule “corrigeGraphe” appelle le graphe “corrigerGraphe”.

2.4 Perspectives autour de la construction des graphes

Dans cette section sont développées des propositions qui n'ont pas été réalisées et expérimentées, contrairement aux idées exposées jusqu'ici. D'abord nous présentons des opérateurs de restructuration des graphes qui paraissent utiles d'après notre expérience concrète de concepteur de graphes. Ces opérateurs sont présentés avec leurs conditions d'application, et parfois, lorsque ceux-ci ont pu être identifiés, avec les contextes dans lesquels leur application semble a priori pertinente.

Dans un deuxième temps, nous nous pencherons sur le problème de la réutilisation des graphes et des morceaux de graphes. La réutilisation est un

problème clé en acquisition des connaissances ([Puerta, Egar, Tu et Musen, 1992], [Klinker, Bhola, Dallemagne, Marques et MacDermott, 1991]): comment profiter des connaissances déjà acquises pour éviter de constamment reconstruire ou "réapprendre" des éléments qui ont déjà été appris, mais pour des situations différentes ? La méthodologie proposée dans la section 2.3.2 n'ignore pas complètement la réutilisation des nodules déjà existants, puisque les connaissances déjà exprimées dans la partie commune du graphe courant sont d'une certaine manière utilisées et généralisées à la nouvelle situation concrète. Mais cela ne concerne que les nodules proches du nodule initial. Appliquée à la lettre, cette méthodologie peut très bien amener le concepteur à créer plus loin dans le graphe plusieurs sous-parties pourtant à chaque fois identiques. Nous chercherons à nous inspirer des travaux en analogie dérivationnelle pour proposer des pistes remédiant à ce problème.

Encore une fois, contrairement à la section 2.3.2, tout ceci ne provient pas d'une pratique, et reste donc prospectif.

2.4.1 Opérateurs de restructuration

Nous définissons les opérateurs de restructuration du graphe comme des opérateurs qui modifient les objets d'un graphe, sans changer le "comportement" du graphe[11].

Deux opérateurs de restructuration semblent s'imposer naturellement. Ils sont complémentaires :

- le premier opérateur regroupe deux branches de graphe en une seule, c'est l'opérateur de **factorisation**.

- le second opérateur est l'opérateur inverse. L'opérateur d'**éclatement** divise une branche en deux.

La figure 2.13 schématise ces opérateurs.

[11] Nous introduirons dans le chapitre 3 la notion de couverture d'un graphe. Nous pourrons dire alors qu'une restructuration ne modifie pas la couverture du graphe.

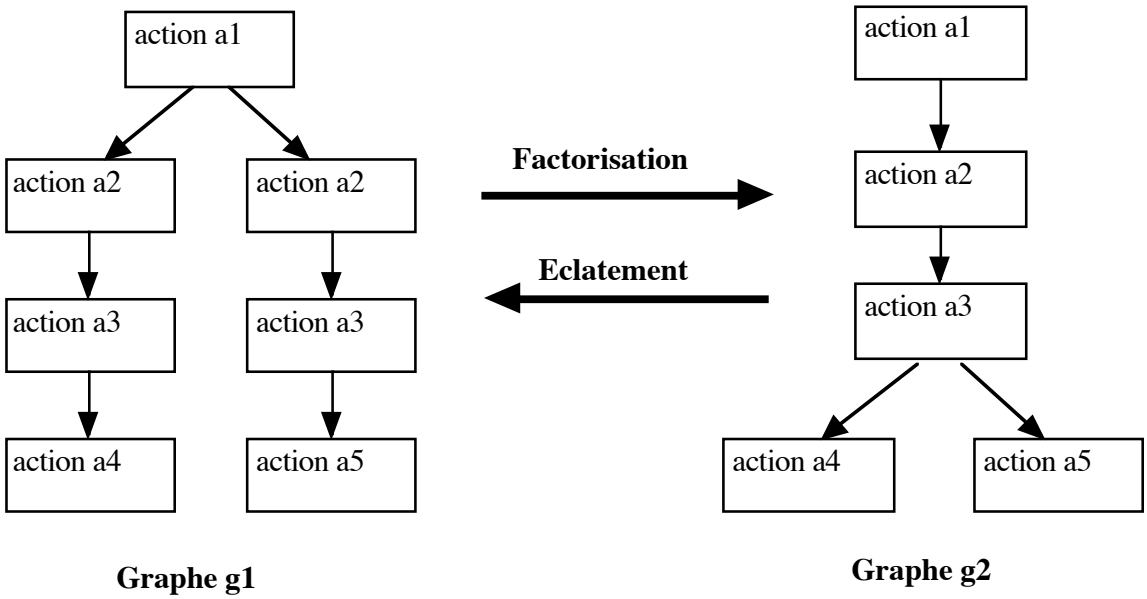


Figure 2.13 : Les deux opérateurs de restructuration

a) La factorisation

La factorisation consiste à regrouper, dans une branche commune, deux branches identiques d'un graphe. Une extension de cet opérateur est la création d'un sous-graphe correspondant à la branche commune. Cette extension a l'avantage de permettre de factoriser un grand nombre de branches sans trop modifier le graphe initial. La figure 2.14 est un exemple de ce type de factorisation avec création de sous-graphe. Le graphe g3 garde la même structure que le graphe initial g1 de la figure 2.13, contrairement au graphe g2. Chaque branche factorisée est simplement remplacée par un nodule qui appelle le nouveau sous-graphe g4.

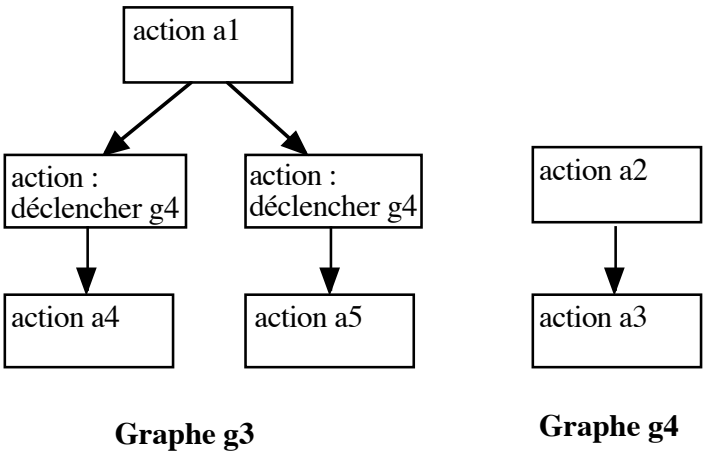


Figure 2.14 : Factorisation avec création de sous-graphe

Nous considérons schématiquement que deux branches sont identiques si elles effectuent les mêmes séquences d'actions dans les mêmes types de situation. On dira dans ces cas là que les deux branches ont un "comportement" identique. Les deux nodules effectuant l'action a2 dans le graphe g1 ne possèdent pas obligatoirement les mêmes règles de choix l'un et l'autre. Mais l'opérateur de factorisation doit faire en sorte de combiner ces règles de choix au sein du nodule effectuant l'action a2 dans le graphe g2 de manière à ce que le comportement des graphes g1 et g2 soit identique.

On peut voir l'intégration de ces opérateurs de factorisation au sein de l'environnement EDINOS de la manière suivante :

- A la demande de l'utilisateur, EDINOS devrait être capable de détecter les branches de graphe "factorisables", c'est-à-dire ayant un comportement commun.
- Lorsque l'utilisateur choisit une branche à factoriser, il spécifie s'il désire créer un sous-graphe spécifique ou non,
- EDINOS devrait alors être capable d'effectuer automatiquement la factorisation en créant les nodules avec les actions appropriées et en corrigeant toutes les règles de choix de manière à conserver strictement le comportement du graphe[12].
- L'utilisateur, ainsi déchargé des soucis de correction des règles de choix, se concentre sur la dénomination des nouveaux nodules de la branche commune. En effet, les anciens nodules possédaient des noms appropriés à leur position dans le graphe. Il est fort possible que les nodules de la branche nouvellement créée aient un sens plus général que les anciens : la tâche de (re)dénomination est du ressort du concepteur (c'est la seule tâche qui lui reste à effectuer, en plus de la décision d'effectuer la factorisation...).

L'intérêt de la factorisation est essentiellement d'améliorer la compréhensibilité des graphes en les rendant plus concis. La factorisation avec création de sous-graphe amène l'avantage supplémentaire de permettre la réutilisation du savoir-faire représenté dans le sous-graphe (section 2.1.3.2).

b) L'éclatement

L'opérateur d'éclatement fait l'opération inverse. Mais il présente un intérêt qui dépasse le simple intérêt de permettre la réversibilité des opérations de

[12] C'est parce que ce type de correction automatique a déjà été développé avec succès au sein de notre outil de révision coopérative (cf. chapitre 3) que nous pensons que ces opérations sont automatisables.

factorisation. En effet, nous avons vu dans la section 2.3.1 qu'un nodule représentait plusieurs situations concrètes différentes. Ceci a pour conséquence qu'un nodule est toujours menacé par le risque d'être trop général : le concepteur peut à tout moment découvrir une situation concrète "mal" représentée par le nodule. Il est important qu'à cet instant le concepteur ait la possibilité de *distinguer* cette situation concrète des autres représentées par le nodule, c'est-à-dire de créer une branche spécifique pour la situation concrète.

Si un opérateur est disponible dans l'environnement EDINOS pour éclater automatiquement une branche en deux, l'utilisateur hésitera peu à effectuer cette opération (et ce d'autant plus si l'opération est réversible). Or il est essentiel de permettre à l'utilisateur de passer facilement d'un doute ou d'une différentiation conceptuelle à une différenciation graphique, au niveau du formalisme de représentation des connaissances. L'environnement joue alors un rôle de support cognitif et la branche dupliquée est la représentation d'une hypothèse qui peut être formulée ainsi : "je crois avoir identifié un cas particulier, qu'il ne faut peut-être pas traiter de la même manière que les autres cas de cette branche". Une fois l'éclatement effectué, l'utilisateur sera mieux en mesure d'identifier les particularités de la nouvelle branche. Si effectivement le cas devait être vraiment distingué, d'autres indices seront alors probablement mis à jour : des actions effectuées inutilement dans la branche (la nouvelle branche pourra être donc raccourcie), de nouvelles alternatives à considérer tout au long de la branche... Toutes ces opérations seront d'autant plus faciles à mettre en oeuvre qu'elles seront effectuées uniquement sur la nouvelle branche et donc n'ont aucune répercussion pour les situations de la branche "mère".

Finalement l'intérêt de cet opérateur d'éclatement vient de ce qu'il est un support important à la recommandation générale suivante qu'on peut donner au concepteur : "*ne pas hésiter à éclater les branches*". L'identification d'une exception à une branche finit souvent par donner lieu au développement d'une branche alternative fournie. L'exception qu'on ose à peine considérer au départ, parce qu'elle est un grain de sable dans l'organisation qu'on avait construite (la branche initiale), peut s'avérer être l'arbre qui cache la forêt, en l'occurrence un ensemble de situations qu'il est important d'expliciter et de différencier... Permettre de bien discriminer les situations fait partie de l'intérêt de la représentation des connaissances par nodules de situation. C'était d'ailleurs dans nos objectifs de trouver un modèle qui permette d'exprimer ce caractère localisé et spécialisé des savoir-faire. Il est donc inutile de chercher à entretenir une branche unique, et de s'efforcer de regrouper les situations concrètes au sein des mêmes objets.

Concrètement, pour lancer l'opérateur d'éclatement, il faut idéalement au départ une situation courante et un nodule. Le scénario serait le suivant :

- Le concepteur utilise l'outil de simulation et détecte une situation courante pour laquelle il doute de la pertinence de son rattachement à la branche courante. Autrement dit, la situation courante ne "cadre" pas avec les autres situations de la branche.

- Le concepteur choisit dans la branche un nodule <nodBranch> à partir duquel il souhaite faire l'éclatement.

- EDINOS duplique la branche. Il devrait être capable de corriger automatiquement les règles de choix du nodule <nodBranch> de manière à aiguiller sur la nouvelle branche pour la situation courante uniquement.

- Comme pour l'opérateur de factorisation, la tâche de l'utilisateur se réduit alors à la (re)dénomination des nodules de la nouvelle branche.

L'opérateur d'éclatement pourra aussi être utile dans un deuxième scénario. Dans ce scénario, l'utilisateur modifie le graphe sans partir d'une simulation. Il se contente de simplement choisir un nodule <nodBranch> à partir duquel il souhaite faire l'éclatement. EDINOS duplique la branche et l'utilisateur doit ensuite modifier "à la main" les règles du nodule <nodBranch> ainsi que la nouvelle branche.

Soulignons enfin que l'éclatement des branches présente aussi des inconvénients. Imaginons une branche éclatée en deux. Les deux branches gardent un certain nombre de caractéristiques communes. Lorsque l'utilisateur découvrira plus tard une étape importante à considérer dans l'une des deux branches, on ne peut pas être certain qu'il se remémorera l'existence d'une branche ressemblante et qu'il pensera alors à envisager également la correction de la deuxième branche...

On touche là au problème général de la réutilisation des branches. C'est notre deuxième perspective de travail, après les opérateurs de restructuration.

2.4.2 Analogie et réutilisation

"La recherche récente en IA et en conception a mis l'accent sur la notion de réutilisation des procédés plutôt que sur le produit de la conception"

[Mostow, 1990, p. 120]

Réutiliser des composants de la BC présente trois avantages :

1) une économie en effort de modélisation, à condition que le coût de recherche puis d'adaptation des composants déjà existants ne soit pas élevé pour le concepteur.

2) l'amélioration de la lisibilité de la BC, puisque la réutilisation aboutit à la définition de sous-graphes communs, et contribue ainsi à réduire la taille de la BC.

3) une économie en effort de maintenance. L'exemple que nous venons de donner à la fin de la section précédente illustre cet aspect (une seule correction de la branche commune d'un côté, de multiples corrections des branches “éclatées” de l'autre).

Le problème de la réutilisation peut être défini de la manière suivante : au cours de la construction (ou de la modification) d'une branche, comment s'inspirer et tirer parti au mieux des connaissances déjà représentées (sous forme de nodules) dans d'autres branches de graphes ? Autrement dit, **comment détecter des situations analogues à celle en cours de modélisation,** qui permettraient éventuellement de guider la modélisation courante ? Le cas limite et le plus intéressant pour le concepteur serait de découvrir des morceaux de graphes déjà définis qui correspondent exactement au problème courant (réutilisation “brute”, sans adaptation).

Nous sommes donc confrontés à un problème d'analogie au sens où l'IA l'entend et que Jaime Carbonell définit de la manière suivante :

“La résolution de problème par analogie consiste à transférer les connaissances des épisodes de résolution de problème passés vers de nouveaux problèmes qui partagent des aspects significatifs avec l'expérience passée correspondante, et à utiliser les connaissances transférées pour construire des solutions pour les nouveaux problèmes”

[Carbonell, 1986, p. 374]

Le raisonnement analogique classique s'appuie donc sur la détection de similarités entre une expérience déjà connue (la source) et un problème nouveau (la cible) pour enrichir la cible (voire fournir des solutions complètes au problème courant) par des éléments provenant de cette source.

“L'hypothèse sous-jacente est que la plausibilité et la valeur de l'analogie est d'autant plus grande que le degré de similarité entre la source et la cible est important”

[Cornuéjols, 1996, p. 236]

Dans notre cadre de modélisation de savoir-faire, nous utilisons le raisonnement analogique comme une aide à la conception, comme un moyen de construire des propositions, des suggestions au concepteur.

Parmi tous les travaux sur le raisonnement par analogie en IA, une branche particulière s'est penchée sur le problème de l'analogie entre "traces de raisonnement" (appelées aussi *dérivations*) : il s'agit de l'analogie "dérivationnelle", introduite par [Carbonell, 1986]. L'analogie dérivationnelle se distingue de l'analogie purement "transformationnelle" parce que c'est une méthode qui reconstruit pas à pas le raisonnement, en reconsidérant chaque décision à la lumière des caractéristiques de la nouvelle situation :

"(.../...) l'analogie dérivationnelle est une méthode reconstructive dans laquelle les lignes de raisonnement sont transférées et adaptées à de nouveaux problèmes [Carbonell, 1986], et pas uniquement les solutions "

[Veloso, 1994, p. 35]

"La méthode de l'analogie dérivationnelle passe en revue les différents pas de raisonnement dans la construction de la solution passée et demande s'ils sont toujours appropriés dans la nouvelle situation ou s'ils devraient être reconsidérés à la lumière de différences significatives entre les deux situations"

[Carbonell, 1986, p. 378]

Ce qui caractérise l'analogie dérivationnelle, c'est donc la capacité de "rejouer" une dérivation. Cela signifie qu'il faut être capable de rapatrier la trace *entière* du raisonnement passé, y compris les justifications des choix effectués à chaque pas (alternatives envisagées, tentatives avortées, etc) :

"Le parti-pris est ici que les cas doivent contenir le raisonnement utilisé pour fournir une réponse, ainsi que les dépendances avec les circonstances particulières du problème, des pointeurs vers les données qui se montrées utiles, une liste de chemins de raisonnement alternatifs non pris, et les tentatives ayant échoué (couplées à la fois avec les raisons de l'échec et les raisons de les avoir tenté)"

[Carbonell, 1986, p. 381]

Cette approche fait écho à la vision "constructiviste" de la résolution de problème vue comme transformation d'un modèle du cas que nous avons présenté dans la section 1.2.1.3. L'analogie dérivationnelle suggère de stocker (afin d'ensuite retrouver et adapter) des dérivations, c'est-à-dire en fin de compte des **enchaînements de modèles du cas** successifs.

Un graphe de nodules de situations est une structure complètement adaptée à l'analogie dérivationnelle dans la mesure où :

- un graphe de nodules est en fin de compte un ensemble de dérivations. L'objet sur lequel Jaime Carbonell applique l'analogie dérivationnelle est un plan de résolution de problème, représenté comme une structure hiérarchique de buts se décomposant en sous-buts : cet objet est équivalent à un graphe de nodules.

- un nodule de situation, étape décisionnelle au cours de la résolution de problème, contient non seulement la liste des alternatives possibles (les nodules successeurs possibles), mais surtout les connaissances pour effectuer les choix pertinents parmi ces alternatives (les règles de choix).

Dans ces conditions, les travaux de J. Carbonell constituent une source d'inspiration très appréciable pour attaquer le problème de la réutilisation. Comme le modèle de l'analogie dérivationnelle de J. Carbonell est assez général, nous dirons que nous appliquons ce modèle aux nodules de situation : nous n'en retiendrons que les aspects qui nous intéressent directement.

Concrètement nous distinguerons deux applications de l'analogie pour l'aide à la construction de graphes de nodules. Une première forme, qu'on appellera "analogie globale", correspond exactement à l'analogie dérivationnelle telle que définie par J. Carbonell. La deuxième forme, qu'on appellera analogie locale, est une aide au concepteur beaucoup plus triviale qui consiste à rechercher toutes les situations exécutant la même action.

a) L'analogie globale

"... les traces du raisonnement passé (appelées dérivations) sont rapatriées si leur segment initial correspond aux premières étapes de l'analyse du problème présent"

[Carbonell, 1986, p. 380]

Des formes d'analogie globale peuvent être esquissées lorsque la future branche à construire commence à se dessiner : l'analogie peut être construite sur la base de **suites d'actions**.

Le schéma envisagé est le suivant :

1) Le concepteur débute la construction d'un graphe, en créant des nodules et en attachant des actions à ces nodules (étape c1 présenté à la section 2.3.2)

2) Au cours de cette étape et avant d'avoir défini la chaîne complète menant du nodule initial au nodule final (étape c2), le concepteur peut avoir recours à un outil de "recherche analogique[13]" qui compare la chaîne de nodules construite au

[13] Cet outil de recherche analogique se distingue de l'outil de factorisation décrit dans la section 2.4.1 par sa souplesse (fuzziness) dans la recherche. Les branches recherchées sont ici similaires, non pas strictement identiques, car la branche initiale est considérée comme une première esquisse. Par exemple l'ordre des actions de la branche initiale ne doit pas être considéré comme définitif : la recherche

reste de la BC. Dans un premier temps, nous considérons que cette mise en correspondance pourra se faire à partir du seul critère de la séquence des actions définie par la branche. La phase de recherche consisterait à trouver des branches proposant des séquences d'actions proches ou identiques à la branche cible.

3) Dans le cas où l'outil de recherche identifie des branches ressemblantes, celles-ci sont tour à tour montrées à l'utilisateur. Celui-ci décide alors ou non de s'inspirer de la branche analogue pour enrichir la branche cible, c'est-à-dire surtout pour créer des nodules manquants et pour compléter les règles de choix des nodules déjà créés. Si la branche identifiée par l'outil de recherche correspond vraiment à la situation courante, l'utilisateur peut choisir de créer un sous-graphe commun.

En résumé, l'analogie *globale* compare la structure des graphes en s'appuyant dans un premier temps sur un critère unique : les actions des nodules. Ce premier critère pourrait ensuite être enrichi pour prendre en compte également des analogies entre les règles de choix. Mais il est prématuré d'approfondir tout ceci sans expérimentations concrètes.

On peut toutefois revenir ici un instant sur les notions de *tâche* et de *méthode*. En effet la réutilisation est au coeur de la notion de méthode. Jusqu'à présent, nous n'avions introduit dans notre modèle que la notion de tâche, assimilée à l'objet “graphe de nodules de situation”. Or l'idée développée dans cette section de rechercher dans l'ensemble de la BC, c'est-à-dire parmi tous les graphes, les branches similaires à la branche courante trouve ses limites lorsque la BC est de grande taille et que plusieurs branches quasiment identiques ont déjà été définies. Quelle branche choisir dans ce contexte ? L'idéal serait que l'outil de recherche analogique identifie en premier une branche unique, pré-vue pour servir de modèle, de source, avant de montrer au concepteur l'ensemble des branches qui sont des copies ou des adaptations de ce modèle. C'est ainsi que nous arrivons à l'idée de constituer à part une base de *méthodes*, vues comme des branches de graphes génériques, destinées à servir de modèles. L'outil de recherche analogique ferait ses mises en correspondance d'abord dans la base de méthodes, avant de les faire dans la base de graphes / tâches. Dans ces conditions, une méthode serait alors un graphe un peu particulier qui contiendrait des informations utiles au concepteur pour son utilisation et son adaptation à la situation courante.

---

analogique doit aller rechercher également toutes les branches contenant les mêmes actions dans un ordre différent.

b) L'analogie locale

L'analogie locale est une forme très réduite d'analogie globale, dont la réalisation est tellement triviale qu'elle est en fait à l'heure actuelle déjà intégrée à l'outil EDINOS, par le biais de la fonctionnalité de recherche des références à un objet donné (présentée dans la section 2.2.1.1).

L'analogie locale, par opposition à l'analogie globale, ne nécessite que de très peu d'informations pour être utilisée. Elle est utile au concepteur au tout début du processus de modélisation d'une situation courante : lorsque celui-ci a identifié l'action adéquate finale à exécuter dans la situation courante. L'analogie locale consiste à rechercher tous les nodules exécutant l'action en question. L'idée est donc de montrer au concepteur toutes les branches aboutissant à la même action que l'action courante. L'intérêt de présenter ces branches au concepteur est double :

- éventuellement éviter au concepteur de véritablement démarrer le processus de modélisation tandis qu'une branche déjà définie correspond exactement à la tâche courante.

- si le concepteur connaît mal la BC et les objets qu'elle contient, les branches proposées peuvent également servir à donner une première idée de la manière dont sont organisés les graphes, quelles actions ont déjà définies, etc. Les branches proposées par l'analogie locale sont d'autant plus intéressantes pour le concepteur qu'elles sont relativement proches conceptuellement de la situation courante (puisqu'elles aboutissent à la même action)[14].

2.5 Bilan et premières évaluations

Dans cette section nous allons tirer deux types de bilans :

- tirer d'abord un bilan général sur le projet “Serveur télématique à destination des affiliés des C.P.R.”.

- ensuite tenter d'évaluer le modèle de représentation des connaissances, notamment à travers l'expérience de cette application au sein des C.P.R.

2.5.1 Bilan du projet de développement d'un serveur télématique

Le serveur télématique, qui a été baptisé “3614 CPRSNCF”, est en service depuis le 1er mai 1996. Sa fréquentation est assez stable et se situe en moyenne

[14] D'autres branches remplissent un rôle similaire : ce sont les branches du graphe au sein duquel la situation courante doit s'intégrer, lorsque ce graphe existe déjà.

autour de 700 à 900 connexions par mois. La population des visiteurs du “3614 CPRSNCF” est constituée pour les deux tiers de retraités, qui utilisent avant tout le serveur pour les services de demande de documents. Une trentaine de questions sont posées chaque mois via le service de messagerie : une vingtaine environ à destination d'un bureau de la Caisse de Prévoyance, une dizaine à destination d'un bureau de la Caisse des Retraites. La figure 2.15 affiche le menu principal du service “3614 CPRSNCF”.

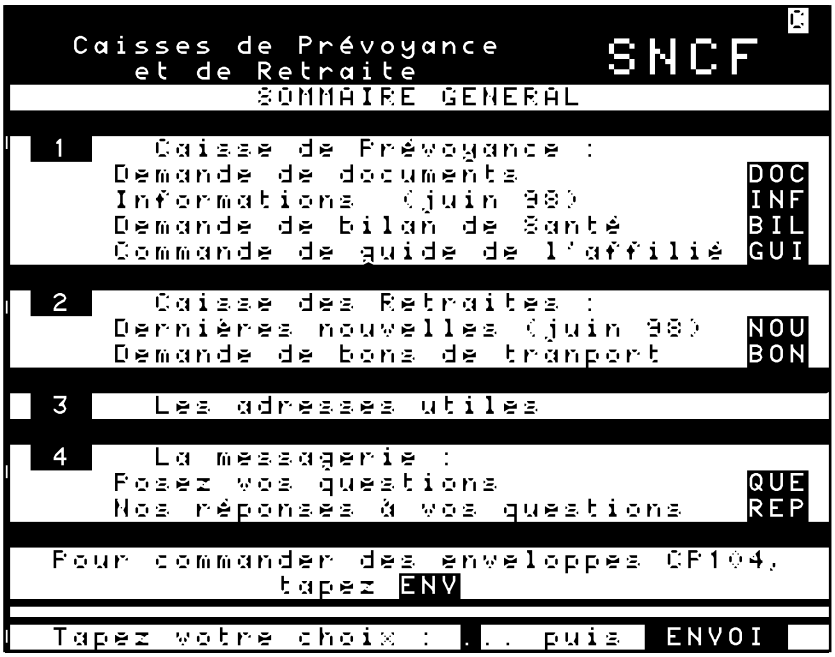


Figure 2.15 : Menu principal du serveur télématique

Parmi les quatre types de services envisagés initialement (cf. section 2.1.1), trois sont complètement ou partiellement ouverts. Ainsi, les demandes de documents et la messagerie sont en service depuis l'ouverture du serveur. La mise à disposition d'informations générales est aussi en place à travers les services “*Informations de la Caisse de Prévoyance*”, “*Dernières nouvelles de la Caisse des Retraites*” et “*Adresses utiles*”. Le service de consultation des dossiers personnels (fiches de décompte), qui nécessite la mise en place de codes confidentiels, a été ouvert à l'automne 1998. La mise à disposition sur le serveur télématique d'informations équivalentes au contenu des guides de l'affilié et du retraité n'a pas été jugée prioritaire. Toutefois, de manière expérimentale, quelques sous-parties du guide du retraité ont été traduites sous la forme de dialogues Minitel .

EDINOS a permis de construire la très grande majorité des dialogues du serveur “3614 CPRSNCF”. Comme nous l'avons montré dans la section 2.2.3, le

dialogue qui suit le choix "Posez vos questions" a été défini sous la forme d'un graphe de nodules de situation (graphe “correspondance” de la figure 2.3). De même, les dialogues des services suivants ont été définis à l'aide d'EDINOS :

1) Demande de documents,
2) Informations de la Caisse de Prévoyance,
3) Demande de bilan de santé,
4) Commande du guide de l'affilié,
5) Dernières nouvelles de la Caisse des Retraites,
6) Adresses utiles.

L'annexe 2 donne le détail du graphe “demDoc” (demande de documents), tandis que l'annexe 3 donne un aperçu de la taille et de la forme des BCs qui correspondent à ces différents services. Nous reviendrons sur l'analyse du contenu de la BC dans cette section.

Si l'on prend comme définition de dialogue personnalisé “dialogue qui prend en compte le profil du minitéliste”, les dialogues 1) et 3), de même que le dialogue “Posez vos questions”, sont véritablement personnalisés. Un exemple d'information utile à la personnalisation du dialogue est le numéro de département de résidence du minitéliste. Il s'est avéré que le recours à une représentation qui permet facilement de proposer des dialogues personnalisés, était vraiment utile pour rendre plus conviviaux les services prévus initialement.

Concrètement, l'intérêt d'avoir utilisé EDINOS pour mettre en oeuvre ces différents services est apparu principalement :

- pour la maintenance des services 2) et 5). Les informations données dans ces branches sont mises à jour régulièrement (au moins mensuellement). De même, le dialogue de la base “correspondance” doit suivre l'évolution de l'organisation interne des C.P.R.

- pour le développement incrémental des services 1) et 3). Lorsque le domaine est relativement complexe, l'approche méthodologique définie dans la section 2.3 est utile pour intégrer progressivement tous les cas de figure à prendre en compte.

En conclusion, EDINOS et la représentation par nodules de situation ont été au coeur du projet “Serveur télématique à destination des affiliés”. Seuls les services de lecture des réponses aux messages et de saisie de bons de transport n'ont pas été développés avec EDINOS. Ces deux services s'apparentent en effet ou bien à la lecture du contenu d'une base de données, ou bien à la saisie d'un formulaire. L'utilisation de l'outil EDINOS a beaucoup influé sur le développement général du projet parce qu'il permettait :

- de proposer très facilement des maquettes de dialogues qui montraient l'intérêt et la faisabilité de nouveaux services,

- de construire et maintenir rapidement et aisément ces nouveaux services.

2.5.2 Évaluation du modèle de représentation des connaissances

Si l'on peut considérer que le bilan du projet "3614 CPRSNCF" est positif et qu'EDINOS participe à cette réussite, il est également important de profiter de cette expérience pour chercher à évaluer le modèle de représentation des connaissances proprement dit. Mais cette tâche d'évaluation est limitée intrinsèquement par la taille de l'expérience effectuée.

Une première manière d'évaluer le modèle proposé consiste à étudier en détail la BC construite dans le cadre du projet. Une seconde manière est de chercher à évaluer la satisfaction des utilisateurs d'EDINOS et à tenter de cerner comment ils réussissent à s'approprier le modèle de représentation des connaissances par nodules de situation. Enfin une discussion générale tentera de dégager un bilan relativement aux objectifs définis au chapitre 1.

2.5.2.1 Analyse du contenu de la BC

L'annexe 3 donne des informations sur le nombre de nodules créés et sur le nombre d'actions de type "demander", "évaluer" et "déclencher".

Ainsi, 287 nodules ont été créés dans 13 graphes. La moyenne est d'environ vingt nodules par graphe créé. Cela va d'un seul nodule créé pour le graphe "commandeGuide" aux 123 nodules du graphe "bilanSanté" qui permettent d'afficher les adresses des 85 centres d'examen de santé disponibles sur le territoire métropolitain.

Sept appels à un autre graphe (action "déclencher") ont été effectués. Un graphe peut ainsi donc être véritablement utile à plusieurs BCs distinctes, comme le graphe "bilanSanté", qui est appelé spécifiquement pour les demandes de bilans de santé d'une part, et pour les adresses utiles d'autre part. Une BC de grande taille peut aussi se partitionner assez naturellement en plusieurs sous-graphes qui correspondent à des sous-domaines, sans que les sous-graphes ne soient appelés plusieurs fois (voir par exemple la base "correspondance" en annexe 3).

Comme cela pouvait être attendu, la plupart des actions qui ont été définies sont des actions de type "demander" (84%). Les écrans définis sont très peu réutilisés : un écran est défini en général pour un seul nodule de situation. En

revanche, les procédures, beaucoup moins nombreuses, sont en moyenne appelées plus de deux fois (au total 41 appels de 17 procédures). Notons au passage que six procédures attachées au total ont été définies.

En conclusion, il semble qu'il soit assez facile de maintenir plusieurs centaines de nodules de situation répartis en plusieurs bases, elles-mêmes constituées éventuellement de plusieurs graphes. Le principe de "modularité médiane" défini dans la section 2.1.2.2 s'avère pertinent puisque les actions de type "déclencher" ont effectivement été utilisées.

2.5.2.2 Évaluation par les utilisateurs

La BC présentée plus haut a été essentiellement définie par l'auteur à l'aide d'EDINOS. Pour mettre à jour les informations délivrées par le serveur télématique, d'autres personnes utilisent désormais EDINOS et manipulent les objets de la BC. Pour chercher à évaluer comment ces utilisateurs réussissent à comprendre et s'approprier notre modèle de représentation des connaissances sous forme de nodules de situation, nous avons soumis un questionnaire aux quelques utilisateurs d'EDINOS, mais une étude plus systématique serait nécessaire pour obtenir des résultats plus complets.

Nous distinguons deux types d'utilisateurs d'EDINOS, qui sont représentées à l'heure actuelle par deux personnes au sein des Caisses de Prévoyance et de Retraite :

- le **rédacteur d'écrans**, qui utilise EDINOS pour corriger et créer les contenus des écrans.

- le **concepteur** proprement dit, qui sait comment spécifier et corriger la logique d'enchaînement des écrans.

Le rédacteur se contente de modifier les écrans Minitel qu'il souhaite, et précise au concepteur comment ces écrans doivent être reliés. Le concepteur crée et corrige les nodules de situation en conséquence. EDINOS est ainsi utilisé de manière partielle[15] par le rédacteur qui ne modifie qu'une partie de la base des actions : la base des écrans.

Le concepteur actuel est l'informaticien qui concevait et gérait les serveurs télématiques avant le développement d'EDINOS. Il utilise EDINOS régulièrement, environ une fois par mois. Il souligne qu'il apprécie l'utilisation d'EDINOS et de la

[15] mais régulière, puisqu'une mise à jour a lieu en moyenne tous les mois.

représentation par nodules de situation parce que cela facilite la maintenance des dialogues Minitel et procure un gain de temps non négligeable ([Panasiuk, 1998]). L'outil employé auparavant[16] pour construire les arborescences des écrans Minitel ne fournissait pas la même souplesse d'utilisation, ni la même puissance d'expression. De plus, l'apprentissage de cet outil requiert des connaissances techniques, notamment la connaissance du langage C, qui le rend inutilisable par un non informaticien.

Le modèle de raisonnement proposé dans EDINOS fournit un cadre simple, mais puissant, qui fait que :

- D'une manière générale, EDINOS est apprécié pour sa "simplicité et son efficacité". La manipulation de l'outil est facile et la formation d'un utilisateur rapide. Toutes ces caractéristiques données par le concepteur laissent supposer, faute de données supplémentaires, que le modèle de raisonnement "par situation" est assez aisément compris et approprié par les utilisateurs. Le fait que la représentation des connaissances par graphes de nodule de situation présente explicitement les étapes de raisonnement et les parcours complets des raisonnements possibles participent à cette compréhensibilité :

"Le principe de décrire une arborescence par nodules, qui se terminera par un écran, permet de visionner clairement une solution."

[Panasiuk, 1998]

- EDINOS peut être utilisé par un non-informaticien. Le rôle de concepteur, tenu à l'heure actuelle par un informaticien, pourrait aussi être tenu par une personne ayant pour spécialité la communication avec les affiliés. Il est même envisagé que le rédacteur d'écrans, non informaticien, remplisse à terme certaines tâches dédiées jusqu'alors au concepteur (modification et création de nodules de situation).

2.5.2.3 Bilan par rapport aux objectifs du chapitre 1

Par rapport aux objectifs définis dans le chapitre 1, nous sommes à ce point de la présentation de notre travail où nous pouvons dégager un premier bilan. Le chapitre 4 sera entièrement consacré à la présentation et à la comparaison des modèles de représentation des connaissances similaires au nôtre. Mais nous pouvons d'ores et déjà tenter de répondre à la question suivante : le modèle que

16 L'outil COSMOS commercialisé par la société R.I.S. Technologies.

nous avons construit et expérimenté répond-il aux objectifs précis que nous nous étions fixés dans la section 1.1 (voir notamment la figure 1.1) ?

Parmi tous les objectifs recherchés, certains font intrinsèquement partie de notre représentation, et par conséquent ne nécessitent pas de discussion particulière :

- le système est toujours **opérationnel** (il fonctionne et propose des résultats),

- le raisonnement effectué par le système peut être suivi de manière **séquentielle** et être **interrompu** à tout moment.

- le savoir-faire est représenté de manière **uniforme** et **explicite**.

Notre discussion se concentrera finalement sur les questions suivantes :

- la représentation des connaissances par nodules de situation est-elle vraiment **compréhensible** et si oui pourquoi ? A cette question est lié l'objectif de représentation **simple** et **explicite**.

- Cette représentation facilite-t-elle la **révision** et si oui pourquoi ?

- Dans quelle mesure la représentation répond-elle à notre exigence de **précision** ?

Enfin en ce qui concerne notre objectif principal **d'incrémentalité** de l'acquisition des connaissances, nous discuterons dans la section suivante de l'origine et de l'intérêt de cette démarche. Cette discussion ne sera pas de même nature que celle de la présente section car elle portera sur le bien-fondé de notre démarche générale, l'incrémentalité, alors que la question que nous nous posons à présent est la suivante : dans quelle mesure notre modèle satisfait nos objectifs ?

Penchons-nous d'abord sur l'objectif de compréhensibilité. Nous avons déjà mentionné dans la section 1.1.3 qu'il était difficile d'évaluer la compréhensibilité d'une représentation. [Gaines, 1996] propose par exemple des mesures de complexité pour évaluer la compréhensibilité des EDAGs, graphes similaires aux graphes de nodules de situations, que nous présenterons en détail dans le chapitre 4. Cette mesure de complexité prend en compte le nombre total de nœuds, d'arcs, de nœuds terminaux et permet d'évaluer chaque graphe à la lumière d'un critère objectif cherchant à noter la **simplicité** du graphe (de même que les graphes de nodules, pour une performance/compétence donnée, plusieurs graphes peuvent être proposés). [Sommer, 1995] propose le même type de mesures, ainsi que des opérateurs de restructuration pour des bases de règles.

Certes, nous nous intéressons ici à la compréhensibilité du modèle général et non à la compréhensibilité d'un graphe comparativement à un autre : ce type de mesures ne nous est donc pas d'une grande aide. Mais nous avions déjà retenu ce

critère de simplicité pour évaluer la compréhensibilité d'un modèle (et, au-delà, d'une instance de modèle) dans la section 1.1.5.

Un nodule de situation est un objet relativement simple. Composé de deux champs uniquement, son comportement est complètement explicite pour l'utilisateur dès lors que celui-ci peut lire le contenu de ces deux champs. On peut schématiser l'interprétation d'un nodule de situation par un utilisateur de la manière suivante : "L'action renvoie n faits différents. En fonction de cela, les règles de choix distinguent n situations différentes." Notre modèle s'appuie sur ce schéma, ainsi que sur le concept de situation. Il semble en effet qu'intuitivement la notion de situation soit évidente à chacun. Inutile d'expliquer à un futur utilisateur ce qu'est une situation pour qu'il puisse comprendre le modèle. Dire par exemple que l'action à effectuer est liée la situation est une banalité. C'est là un argument en faveur de la compréhensibilité du modèle : nous nous appuyons sur une notion commune et simple, connue de tous. La nouveauté de notre modèle est de proposer de représenter des "chaînes de situations" et d'identifier le raisonnement à ces chaînes de situations.

Deux critiques peuvent être faites relativement aux arguments qui viennent d'être exposés. D'abord la compréhensibilité d'un nodule est étroitement dépendante de l'interface graphique de l'environnement de construction (ou de simple visualisation) de la BC. A l'heure actuelle, l'affichage d'un graphe dans EDINOS ne propose dans un premier temps que les noms des nodules du graphe. Le contenu d'un nodule (ses champs) est affiché dans des fenêtres spécifiques, ouvertes dans un second temps. D'autres moyens pourraient être trouvés pour accélérer ou faciliter l'accès à ces informations (par exemple afficher des graphes dont les noms contiennent les noms des actions en plus des noms de nodules). La seconde critique concerne la compréhensibilité du modèle et de la notion de situation. La distinction entre situation courante et nodule de situation a déjà été détaillée dans la section 2.3.1. Il semble que cette différence, incontournable, de niveaux d'abstraction soit la principale difficulté lors de "l'apprentissage du modèle" par un novice. On peut résumer ainsi cette difficulté, tout de même assez surmontable : il faut toujours se rappeler que le nodule de situation peut représenter également d'autres situations concrètes que la situation courante ou la situation concrète qu'on a en tête au moment présent.

En plus de la simplicité, une deuxième caractéristique est étroitement liée à la compréhensibilité : il s'agit de la transparence, de la capacité de la représentation à être **explicite**. Si l'objet "nodule de situation" est explicite, là encore l'interface de l'environnement informatique joue un rôle déterminant pour développer ou non cet aspect explicite. Nous avons vu en effet dans la section 2.2.1.1 lorsque nous avons

insisté sur l'importance de la "compréhension par exploration" que la compréhension d'un objet passait par la compréhension des objets qui lui sont liés. EDINOS permet donc de **rendre explicites les liens entre objets** grâce aux outils de recherche des objets faisant référence aux autres.

Une seconde particularité de notre modèle est de représenter explicitement la **trace** des raisonnements : les nodules précédemment activés au cours du fonctionnement du moteur constituent naturellement les étapes du raisonnement. Ils ont été déjà commentés par le concepteur et peuvent être "explorés" en détail par l'utilisateur.

Pour clore ce développement sur la compréhensibilité de notre représentation, on peut résumer l'argument fort en faveur de la compréhensibilité des nodules de situation de la manière suivante : il y a une grande "proximité entre le niveau conceptuel et le niveau implémentation". En effet, le niveau conceptuel est le niveau "modèle" dont nous venons de vanter la simplicité. Le niveau "implémentation" est complètement calqué sur le niveau conceptuel[17] : il contient le langage de définition des actions, des règles de choix. L'intérêt d'avoir un lien si étroit entre modèle et programme, c'est que le passage de l'implémentation au modèle est immédiat pour l'utilisateur.

Cette dernière caractéristique est également un argument important en faveur de la réalisation de notre objectif de **révisabilité**. On trouve le même type d'arguments dans le projet DIVA, où [David, Krivine et Ricard, 1993] parlent de débogage au niveau connaissances (Knowledge Level). A la différence d'un débogage au niveau implémentation :

> *"La correspondance structurelle entre l'implémentation de DIVA et son modèle au niveau Connaissances (.../...) nous a permis de déboguer le système à un niveau beaucoup plus abstrait"*
>
> [David, Krivine et Ricard, 1993, p. 390]

Cette idée a été formulée également par [Reinders et al., 1991] qui ont introduit un **principe de correspondance structurelle** qu'on peut exposer ainsi : l'architecture du système doit préserver la structure du modèle conceptuel correspondant. L'intérêt du principe de correspondance structurelle est d'alléger les problèmes bien connus d'explication, de construction et de maintenance de la BC.

Une deuxième caractéristique importante de notre système s'ajoute à cela si l'on veut démontrer que l'on répond à l'objectif de révisabilité. C'est le fait que notre système est un **système apprenti** (voir section 1.2.2.2). Nous avons présenté

[17] A tel point qu'on peut avoir du mal à différencier les deux niveaux...

dans ce chapitre 2 uniquement l'outil de construction de la BC EDINOS, en insistant beaucoup sur son module de simulation. Nous développerons dans le chapitre suivant l'outil de révision coopérative REVINOS, couche supplémentaire au dessus d'EDINOS, qui fait de notre système un véritable système apprenti permettant de faire de l'acquisition de connaissances et de la révision en contexte. Nous présenterons alors les fonctionnalités que nous avons développées pour faciliter la révision et nous montrerons comment la représentation par nodules de situation est intéressante pour ce processus de révision.

L'exigence de **précision** est le dernier objectif qu'il nous reste à évaluer. Nous avions défini cette exigence de précision en insistant sur l'importance d'éviter la surgénéralisation. Notre approche incrémentale, de type ascendante, conduit en effet à constamment chercher à construire des généralisations. Nous souhaitions une représentation qui permette d'exprimer autant de cas particuliers que souhaités. Or toute situation inédite peut effectivement être représentée par un nouveau nodule qu'on accroche à un graphe ; le concepteur a toute liberté de faire cela. De plus, les nodules de situation permettent un certain compromis entre cette précision recherchée et la compréhensibilité générale de la BC. Un nodule de situation peut être au niveau d'abstraction souhaité par le concepteur : il peut même correspondre, à la limite, à une unique situation concrète. Mais on peut souligner que, même dans ce cas, le parcours menant à un nodule terminal comprend des nodules qui sont activés dans d'autres situations concrètes. Autrement dit, il y a toujours un grand nombre de nodules de la BC qui sont communs à l'ensemble des cas résolus représentés dans la BC. Ces parties communes des graphes maintiennent une compréhensibilité de l'ensemble du graphe, par opposition par exemple à un ensemble de règles indépendantes où chaque prémisse doit définir exhaustivement le cas particulier associé.

L'expressivité de notre modèle de représentation des connaissances trouve ses limites lorsqu'on l'on veut représenter des connaissances de contrôle usuelles où des itérations sont nécessaires. En effet, notre modèle ne prévoit pas de revenir en arrière sur un nodule préalablement activé : nous l'avions limité à des raisonnements monotones (voir section 1.3.5.2). Toutefois il apparaît possible d'effectuer cette extension assez simplement, comme nous allons le montrer dans la section suivante.

2.5.3 L'extension aux itérations

L'algorithme du moteur exploitant les graphes de nodules a été donné dans la section 1.3.4. Cet algorithme n'interdit pas a priori de réactiver un nodule qui a déjà été activé. Les boucles dans les graphes ne sont donc pas spécifiquement interdites dans notre modèle. Toutefois, nous avons fait le choix d'avoir une MT globale[18], transparente, qui peut être interrogée à tout moment : les faits qui seront renvoyés lors du second passage dans la "boucle", vont alors être en concurrence avec ceux renvoyés lors du premier passage. L'extension aux itérations nous oblige donc à renoncer à la monotonie du raisonnement.

Une solution consiste à distinguer l'instruction "si: *prémisse* alorsAller: *noduleN1*" qui apparaît dans les règles de choix, d'une instruction "si: *prémisse* alorsRetourner: *noduleN2*" valide lorsque *noduleN2* fait partie des nodules déjà activés par le moteur. Cette nouvelle instruction provoquerait l'effacement de tous les faits de la MT qui ont été renvoyés par les actions des nodules activés de *noduleN2* jusqu'au nodule courant. Le principal inconvénient à la création de cette nouvelle instruction réside dans les répercussions qu'elle nécessitera sur l'application de révision de la BC (REVINOS), que nous présenterons dans le chapitre 3.

Notons que cette extension permet d'éviter l'adaptation spécifique du modèle pour la touche "sommaire" (cf. section 2.1.3.1).

2.6 Discussion autour de l'approche incrémentale

Avant d'essayer de cerner quels sont les autres domaines d'application possibles pour notre modèle et notre outil, nous voudrions nous arrêter un instant sur le problème de l'incrémentalité en acquisition des connaissances. Nous avons en effet dès le départ axé notre démarche sur cet objectif, sans jamais le remettre en question. A présent que nous avons présenté en détail notre approche, il apparaît intéressant d'en discuter les avantages et des inconvénients, relativement aux autres approches utilisées en acquisition des connaissances.

2.6.1 Modélisation préalable contre approche incrémentale

[18] Par opposition à une MT attachée au nodule et valide uniquement quand il est nodule courant.

Nous avons adopté une approche d'acquisition incrémentale des connaissances, et ce dès la formulation de nos objectifs dans la section 1.1. Désireux de construire des systèmes évolutifs, nous avons immédiatement précisé que nous voulions que le système soit dès le départ opérationnel. Or cette approche incrémentale s'oppose à une approche générale dominante dans le domaine de l'acquisition des connaissances qui consiste à établir préalablement des modèles conceptuels de l'expertise, avant l'étape d'opérationalisation de ces modèles ([Krivine et David, 1991], [Lemaire, Safar et Yonnet, 1996]). Nous appellerons cette approche dans la suite : l'approche "modélisation préalable", dont le modèle KADS est le représentant le plus connu.

2.6.1.1 La modélisation préalable

A l'origine de l'approche modélisation préalable, il y a sans doute l'article historique d'Allen Newell ([Newell, 1982]), qui propose de distinguer un "niveau Connaissances" au dessus du niveau symbolique. L'idée que propose Newell est de revenir à l'essentiel de ce qu'est l'acquisition des connaissances, c'est-à-dire la définition du comportement du système (en termes de buts, de *connaissances*), et donc de s'abstraire de l'influence de l'implémentation et des problèmes "techniques" liés.

"Il est utile de décrire le comportement de résolution de problème avec des modèles plus adéquats, plus abstraits, que ceux qui sont offerts par les langages orientés 'représentation des connaissances'. Ces modèles de haut-niveau de description sont appelés modèles de résolution de problème"

[Karbach, Linster et Voss, 1990]

L'élaboration de "modèles de résolution de problème" indépendamment des soucis d'implémentation est utile parce que:

1) L'activité d'acquisition de connaissances est une activité de modélisation, qui n'est pas triviale, et qu'il faut donc pouvoir valider,

2) Cette activité met en jeu plusieurs experts qui doivent pouvoir se comprendre, et qui ne sont pas forcément informaticiens.

Dans ces conditions, l'acquisition de connaissances a besoin de langages de description pour les modèles de résolution de problème, afin de permettre à différents experts de communiquer sur la base de ces modèles qui seront ensuite validés [Wielenga, Schreiber et Breuker, 1992].

Or un certain nombre de critiques peuvent être faites à l'approche "modélisation sans opérationalisation". La première critique prend son inspiration

dans le génie logiciel. Une des caractéristiques du processus de développement de logiciel est sa nature itérative : certaines étapes déclenchent la révision du résultat des étapes précédentes ([Gaudel, Marre, Schlienger et Bernot, 1996]). On peut supposer que dans le domaine du génie cognitif ou de l'ingénierie des connaissances, le passage de l'étape du modèle conceptuel à l'étape du modèle opérationnel n'échappe pas à cette règle :

"Les systèmes non-opérationnels excluent les effets de l'introspection[19] — qui résultent du retour que fournit un système en marche — dans l'élaboration du modèle courant. Pour ces raisons, les éléments du modèle et le modèle résultant doivent être exécutables"

[Linster, 1993, p. 476]

Cette critique prend d'autant plus de sens lorsque les modèles construits sont des réutilisations de modèles génériques identifiés auparavant. La réutilisation de modèles génériques soulève un certain nombre de problèmes ([Rademakers et Vanwelkenhuysen, 1993]), qui peuvent entraîner la révision du modèle conceptuel. Par exemple :

"Il est possible que les composants génériques choisis au départ du processus de développement du SBC s'avèrent inadéquats ultérieurement. Cette situation peut être due au fait qu'il n'y ait pas suffisamment d'information pour choisir un modèle au début du projet — le problème est d'habitude défini de façon très générale — ou au fait que le modèle soit trop générique pour fournir des critères discriminants efficaces."

[Cañamero, 1995]

Si l'on peut espérer que ce problème puisse se régler lorsque l'on parviendra à fournir des critères fiables et précis pour le choix des composants génériques, il n'en reste pas moins que le processus de conception de SBC donne souvent lieu à des retours en arrière nécessaires pour un bon ajustement du SBC à la réalité. Ces retours en arrière peuvent aussi bien s'avérer nécessaires une fois l'étape d'opérationalisation effectuée. Lorsque ce cas de figure se présente souvent, le découpage "modélisation préalable puis opérationalisation" perd de son intérêt. Nous retrouvons ainsi l'opposition systèmes définitifs / systèmes évolutifs introduite dans la section 1.1.2.

La seconde critique à l'approche "modélisation sans opérationalisation" est de nature pragmatique. Pour certaines expertises peu complexes, le passage obligé par une étape intermédiaire de modélisation dans un langage spécifique — qu'il faut apprendre et maîtriser — alourdit considérablement le processus de construction de BC.

---

19 "insight"

Une alternative se propose donc à la modélisation préalable : opérationaliser immédiatement et enrichir incrémentalement le "champ d'expertise" couvert par le système.

2.6.1.2 Origine de l'approche incrémentale

Il est logique de retrouver dans notre opposition entre modélisation préalable et approche incrémentale une opposition similaire en génie logiciel, entre analyse et prototypage rapide. En effet le génie logiciel, qui propose un ensemble de techniques et de méthodes pour faciliter la construction de logiciels, et le génie cognitif, qui, lui, s'intéresse à la construction de Base de Connaissances, partagent un certain nombre d'objectifs et de problématiques. La recherche de modularité par exemple est commune aux deux champs : faciliter la compréhensibilité et la correction d'un programme ou d'une BC, chercher à définir des composants réutilisables, etc sont des soucis communs.

Si l'on cherche à identifier les sources de l'approche incrémentale que nous utilisons, on peut trouver deux inspirations principales:

- le génie logiciel, et l'approche prototypage,
- l'apprentissage incrémental.

a) Le prototypage rapide

Le prototypage peut être défini de la manière suivante :

"Le prototypage est une approche dans le développement de logiciel qui met l'accent sur la préparation de versions immatures qui peuvent être utilisées comme base pour évaluer les idées et décisions, comme un préalable à la préparation d'une version qui sera considérée comme complète et complètement certifiée (c'est-à-dire considérée comme suffisamment mûre pour livraison au client)"

[Riddle, 1984, p. 20]

Cette définition rejoint complètement le point de vue exposé dans la section 1.1.2 sur la construction incrémentale de systèmes évolutifs ou provisoires. Les avantages du prototypage rapide sont :

- de démontrer la faisabilité technique ou l'efficacité de certaines parties du système,
- de permettre la détection au plus tôt de divergences entre concepteurs et utilisateurs :

"(.../...) le prototypage pour des systèmes applicatifs interactifs sert en général à améliorer la communication entre les développeurs et les utilisateurs en ce qui concerne la pertinence des interfaces personne-machine, ou plus généralement l'adéquation entre les fonctions du système et les tâches de travail"

[Floyd, 1984, p. 3]

On distingue deux types de prototypage [Floyd, 1984] :

- le prototypage "vertical" : seules certaines fonctionnalités sont opérationnelles dans le prototype, mais elles sont dans leur forme finale,

- le prototypage "horizontal" : les fonctionnalités ne sont pas développées en détail comme demandé dans la forme finale.

Si l'on considère un cas résolu, comme une fonctionnalité (très) spécifique (le prototype fonctionne ou non pour le cas résolu donné), le prototypage vertical relève en quelque sorte d'une approche incrémentale.

Mais on peut également trouver une seconde source à l'approche incrémentale d'acquisition des connaissances : c'est l'apprentissage incrémental, qui s'intéresse à la correction et à l'extension de définitions de concepts, au fur et à mesure que des exemples du concept à apprendre sont donnés au système d'apprentissage.

b) l'apprentissage incrémental.

On présente souvent les techniques d'apprentissage comme un moyen de se passer de l'étape d'analyse... Les algorithmes d'apprentissage permettent en effet de construire automatiquement une BC à partir d'un ensemble d'exemples. Ces exemples peuvent être des exemplaires de concepts ou des cas résolus.

Or pour des raisons pragmatiques, il s'avère souvent utile que l'apprentissage puisse se faire de manière incrémentale ou séquentielle. Il n'est effectivement pas toujours possible de disposer d'un ensemble d'exemples déjà disponibles préalablement au lancement de l'algorithme d'apprentissage. Mais surtout, des nouveaux exemples à prendre en compte, sont toujours susceptibles de faire leur apparition au cours de la vie du système : nous avons évoqué dans le chapitre 1 ce problème de l'évolutivité d'un domaine de connaissances.

Ainsi, à partir d'un algorithme d'apprentissage conçu au départ pour fonctionner de manière "batch" (par exemple, ID3 [Quinlan, 1986]), il est souvent jugé nécessaire de développer ensuite des versions incrémentales (dans notre exemple, les algorithmes ID4 [Schlimmer et Fischer, 1986] et ID5 [Utgoff, 1989]).

Inversement, d'autres algorithmes ont été conçus dès l'origine pour permettre un apprentissage incrémental, comme COBWEB ([Fisher, 1987]).

Parmi les premiers travaux influents en apprentissage incrémental, on trouve ceux de Patrick Winston qui introduit la notion de "near-miss" ([Winston, 1975]) qu'on peut traduire par "nuance critique" ([Kodratoff et Ganascia, 1986]) :

"Une nuance critique ('near miss') est un exemple négatif qui, pour un nombre limité de raisons, n'est pas une instance de la classe apprise"

[Winston, 1992, p. 351]

Dans l'approche de P. Winston, un modèle "évolutif" est modifié au fur et à mesure que des nuances critiques sont "rencontrées". L'intérêt de ces nuances critiques est de focaliser le système d'apprentissage sur des aspects pertinents permettant de différencier les exemples positifs des exemples négatifs.

P. Winston développe un argumentaire en faveur de l'apprentissage incrémental en précisant que d'une manière générale l'apprentissage doit être fait "par petits pas". Il énonce la "loi de Martin" :

"On ne peut pas apprendre quelque chose à moins de le savoir déjà quasiment"

[Winston, 1992, p. 359]

Pourtant il est exagéré de dire que l'algorithme de P. Winston soit véritablement un algorithme d'apprentissage incrémental. En effet, les "near-miss" jouent un rôle central dans l'algorithme, mais ce ne sont pas vraiment des exemples "naturels" : le micromonde des blocs sur lequel travaille P. Winston n'est pas uniquement constitué d'exemples positifs et de near-miss, il existe également des exemples négatifs qui ne sont pas des near-miss, et que l'algorithme ne sait comment exploiter parce qu'ils sont trop différents du modèle courant... Dans les faits, l'algorithme de P. Winston "pêche" ses near-miss dans une base d'exemples connus au départ.

Or dans la problématique générale de l'apprentissage incrémental il faut être capable d'apprendre les exemples au fur et à mesure qu'ils se présentent, et tels qu'ils se présentent (sans critère de sélection). Aussi l'apprentissage incrémental est plus qu'un simple apprentissage séquentiel :

"L'apprentissage est de type incrémental lorsque l'histoire du système n'est pas indifférente"

[Cornuéjols, 1989, p. 25]

D'après cette définition, si l'ordre de présentation des exemples a des répercussions sur la nature des connaissances apprises, on pourra parler d'apprentissage incrémental. Une caractéristique importante de l'apprentissage incrémental est que l'arrivée d'un nouvel exemple ou d'un nouveau cas doit être intégré à la BC sans disposer d'aucune autre information supplémentaire : si le système doit effectuer certains choix au cours de l'apprentissage, il ne peut disposer d'autre information "extérieure"[20] que l'exemple lui-même. Cela fait partie des difficultés de l'apprentissage incrémental, et ce sont d'ailleurs des problèmes généraux qui touchent tout agent cognitif évoluant dans un environnement. Si le nouvel exemple suscite des remises en question au sein de la BC, des ambiguïtés,... le système n'a pas toujours les moyens d'éclaircir complètement ces questions. Mais il doit pourtant intégrer l'exemple à la BC.

Nous verrons dans le chapitre 3 que dans notre approche nous pouvons éviter ces problèmes spécifiques. Parce que nous nous situons dans un cadre où nous avons un utilisateur "sous la main" et parce que nous nous sommes défini un objectif de *précision* des connaissances représentées (section 1.1.4), notre outil de révision a la possibilité de chercher à éclaircir "en temps réel" les zones d'incertitude apparues au cours de l'apprentissage.

Notons qu'une solution puissante au problème de l'incrémentalité qui ne repose pas sur l'utilisateur, a été proposée par Tom Mitchell sous le nom d'"espace des versions". L'espace des versions est l'ensemble des définitions ou hypothèses qui permettent à l'instant t de rendre compte de tous les exemples et contre-exemples déjà rencontrés. En repoussant ainsi le choix d'une hypothèse spécifique, l'algorithme de T. Mitchell est capable d'apprendre indépendamment de l'ordre des exemples. Deux ensembles de définitions courantes sont donc maintenus simultanément, un pour les définitions maximalement générales, un autre pour les définitions maximalement spécifiquement :

"La représentation de l'espace des versions avec ses ensembles de versions maximalement générales et maximalement spécifiques, apparaît bien adaptée à l'apprentissage de règles à partir d'exemples présentés de manière séquentielle"

[Mitchell, 1977, p. 310]

2.6.1.3 Avantages de l'approche incrémentale

[20] "Extérieure" au sens : autre qu'une théorie du domaine plus ou moins étendue, que la BC elle-même, ou que des préférences pour l'apprentissage données préalablement par l'utilisateur.

Adopter une approche d'acquisition incrémentale des connaissances permet de construire des solutions ex nihilo. Pour que ce processus satisfasse le concepteur de la BC, il est nécessaire que l'intégration d'une nouvelle solution se fasse : facilement et rapidement. Dans ce schéma, il est effectivement primordial qu'un nouveau cas aussitôt détecté, puisse aussitôt être intégré : sinon l'approche incrémentale ne présente pas d'intérêt. Il faut donc un bon système d'apprentissage de solutions complètes pour permettre une bonne souplesse dans le processus d'acquisition des connaissances. Les systèmes de raisonnement à partir de cas sont des bons exemples de systèmes capables d'apprendre instantanément une solution inédite.

Le grand argument en faveur des systèmes capables d'apprendre directement à partir de rien ou de pas grand chose, c'est qu'ils permettent de faire l'économie d'une analyse exhaustive du domaine. Le système donne des solutions sans qu'il ait été nécessaire préalablement de passer en revue toutes les connaissances du champ de compétences qu'on veut donner au système. Certes, à moyen terme le domaine sera aussi couvert complètement, dans la mesure où le champ de compétences du système est étendu au fur et à mesure que de nouvelles solutions sont fournies au système. Mais la souplesse du processus incrémental permet de disposer d'un système opérationnel dès le commencement du processus (avec alors une compétence limitée).

Plutôt qu'en rester à l'opposition entre analyse préalable et apprentissage incrémental, on peut proposer dans la lignée de l'approche GDM ([Terpstra, van Heijst, Shadbolt et Wielenga, 1993]) et du projet KEW ([Shadbolt and Wielenga, 1990]) de **faire progressivement l'analyse, tout en opérationalisant.**

2.6.2 Une modélisation opérationnelle incrémentale ?

Dans l'environnement d'aide à l'acquisition des connaissances KEW, le choix a été fait d'avoir un modèle conceptuel toujours exécutable ([Shadbolt and Wielenga, 1990]). L'approche GDM (pour General Directive Models) voit l'acquisition des connaissances comme un processus cyclique de raffinements successifs de modèles, appelés GDM :

"Le développement du modèle et l'acquisition des connaissances statiques du domaine sont entremêlés. Au départ un modèle hautement abstrait de la tâche de raisonnement est utilisé pour acquérir des connaissances du domaine basiques. Par la suite, le modèle est raffiné en utilisant les propriétés des connaissances du domaine déjà acquises"

[Terpstra, van Heijst, Shadbolt et Wielenga, 1993, p. 452]

C'est l'étroite relation entre tâche effectuée et connaissances du domaine, évoquée dans la section 1.2.1.2, qui a amené les auteurs à proposer cette approche.

"(../...) bien qu'il soit certainement possible d'éliciter les caractéristiques pertinentes de la tâche que le système doit accomplir, en général il ne sera pas possible de choisir un modèle unique sans éliciter les connaissances du domaine"

[Terpstra, van Heijst, Shadbolt et Wielenga, 1993, p. 430]

Cette approche rejoint donc complètement la nôtre puisqu'il s'agit de développer un système toujours opérationnel en faisant l'analyse de manière incrémentale[21].

Récapitulons comment nous sommes arrivé nous aussi à cette idée de modélisation opérationnelle incrémentale, et quelles sont les caractéristiques de notre approche :

- Nous avons suivi depuis le départ un objectif général d'acquisition incrémentale des connaissances. Le fait que la BC devait être opérationnelle a découlé de cet objectif.

- Nous n'avons jamais fait de distinction entre un niveau conceptuel et un niveau opérationnel.

- Notre parti-pris de toujours permettre la révision nous mène finalement à réviser au niveau conceptuel (c'est-à-dire plus explicitement au "niveau situation").

- Nous avons axé l'acquisition des connaissances sur des exemples concrets, reprenant le principe du raisonnement à partir de cas et des systèmes apprentis.

Ce qui semble très intéressant dans cette approche de "modélisation opérationnelle incrémentale", c'est qu'elle permet de dépasser l'opposition que développe Janet Kolodner entre "raisonnement basé sur les modèles" et raisonnement à partir de cas (RPC). Nous avions déjà évoqué cette opposition dans le chapitre 1. L'argument avancé par les promoteurs du RPC est que le RPC peut être utilisé dans des domaines non "modélisés" ou difficilement "modélisables" :

"Le RPC, par opposition, excelle dans la couverture de "domaines à faible théorie", domaines dans lesquels on ne comprend pas encore suffisamment bien les phénomènes pour les enregistrer sans ambiguïtés"

---

[21] Il y a bien une étape d'analyse dans la construction de BC avec des nodules se situation, qui se déroule selon le processus décrit à la section 2.3.2 : cette étape a lieu lors de l'étude de la nouvelle situation, et elle aboutit à la création ou la réutilisation d'objets (situations, actions, faits) de la BC.

[Kolodner, 1993, p. 15]

L'approche "modélisation opérationnelle incrémentale" peut aussi être mise en oeuvre pour de tels domaines. La différence est que contrairement aux systèmes de RPC, cette approche cherchera à *expliciter* (donc rendre compréhensible) progressivement et comparativement les savoir-faire du domaine. Cette démarche est également celle des outils d'élicitation des connaissances comme ETS ([Boose, 1985]), basés sur des techniques de grilles-répertoires et dont nous reparlerons dans le chapitre 3.

Le grand risque de la modélisation incrémentale, surtout lorsqu'elle est guidée par des exemples, c'est de "réinventer la roue". En effet dans le feu de l'exemple à intégrer, il n'est pas toujours facile de prendre le recul nécessaire de "l'analyste". D'où l'enjeu important de fournir des outils suggérant des réutilisations de branches de graphes aux moments opportuns (cf. section 2.4.2).

2.7 Domaines d'application

La question importante à laquelle nous devons maintenant répondre est la suivante : à quelles types de tâches notre modèle est adapté ?

Une première réponse a été donnée dans la section précédente. Lorsque le domaine n'est pas explicité au départ, mal cerné, mal connu, une démarche de modélisation incrémentale peut être intéressante pour construire une BC opérationnelle progressivement : l'approche "nodules de situation" peut alors être pertinente.

Une seconde réponse plus générale peut être formulée à partir des objectifs de départ exposés dans le chapitre 1 : les nodules de situation s'avèrent adaptés à la construction de systèmes évolutifs. En dehors de choix idéologiques, plusieurs circonstances concrètes peuvent mener à adopter une démarche de construction de système évolutif :

1) Seule une partie parfaitement délimitée du savoir-faire est explicitée par les experts. Exemple : on sait uniquement résoudre certains cas.

2) Il a été constaté que le savoir-faire évolue avec le temps.

3) On sait que dans le domaine, le savoir-faire explicité peut toujours être remis en question.

Une approche plus prudente pour répondre à la question "Quand utiliser des nodules de situation ?" consiste à généraliser l'application qui a été faite des nodules de situation à la conception de services télématiques. Le problème général

de notre application est un problème de **classification** où les données sur le cas ne sont pas connues au départ de la tâche.

Faut-il généraliser jusqu'à dire que les nodules de situation sont adaptés à tous les problèmes de classification ? Pour avoir une idée de la réponse à cette question, examinons de quelle manière notre modèle pourrait effectuer une méthode classique de résolution de problème en acquisition des connaissances comme "couvrir-et-différencier", utilisée par le système MOLE ([Eshelman, 1988]) pour faire de la classification.

La méthode couvrir-et-différencier fonctionne de la manière suivante. Une Base de Règles est capable d'établir des liens entre des symptômes et des causes. Pour un ensemble de symptômes donnés, MOLE établit ainsi une liste de causes candidates (qui couvrent les symptômes). A partir de cette liste d'explications candidates, MOLE est capable de différencier les explications candidates en posant des questions à l'utilisateur. Ce processus de différentiation peut être parfaitement exprimé par les nodules de situation, à condition de représenter chaque situation de différentiation par un nodule... A chaque nodule correspondrait alors implicitement une liste d'hypothèses candidates : ces hypothèses candidates, qui sont des informations centrales et explicites dans MOLE, deviendraient annexes, implicites dans le SBC. En revanche, le savoir-faire de différentiation pourrait être, avec une représentation par nodules, complètement localisé, indépendant d'un nodule à un autre, et par conséquent plus précisément exprimé.

Il apparaît aussi que pour des tâches parfaitement précises et connues, les méthodes de résolution de problèmes lorsqu'elles existent fournissent des représentations adaptées et plus efficaces qu'un modèle général comme les nodules de situation.

Prenons un problème de conception pour étayer notre argument. La méthode proposer-puis-réviser utilisée par l'outil SALT [Marcus, 1988] est basée sur un type de raisonnement par contraintes. Trois types de données sont nécessaires à SALT : des valeurs initiales, des contraintes et des remèdes lorsque les contraintes sont violées. Les nodules de situation ne sont pas très adaptés pour vérifier des contraintes dans le sens où la déclaration de ces contraintes n'est pas aisé dans notre modèle. En revanche, les nodules se montrent intéressants pour spécifier l'ordre de vérification des contraintes, ainsi que les actions à entreprendre en fonction des résultats de ces vérifications.

Ici encore, il est possible d'utiliser une représentation par nodules de situation pour modéliser la tâche. Mais le recours à une représentation spécifique (dans notre dernier exemple, pour exprimer de manière déclarative une liste de contraintes) est toujours plus aisé.

On pourrait finalement positionner l'approche nodule de situation comme une proposition de modèle générique pour construire une BC à partir de rien, utile par exemple à des concepteurs qui ne veulent pas se préoccuper de chercher une réutilisation peu rentable si la BC reste de petite taille. Autrement dit, la représentation par nodules de situation pourrait servir de représentation générique, point de départ d'une analyse incrémentale opérationnelle. On peut imaginer que le système, durant cette modélisation opérationnelle, finisse éventuellement par reconnaître (en détectant des similitudes) le type de méthode de résolution de problème dont a besoin le concepteur : il lui proposerait alors de transférer, puis utiliser les représentations et outils d'acquisition des connaissances adaptés à la tâche.

Chapitre 3 :

REVINOS : un outil de révision coopérative autour de la représentation par nodules de situation

Pour compléter EDINOS, l'outil de construction de la BC et de simulation, nous avons développé un outil, baptisé REVINOS[1], dont le but est de guider le processus de révision de la BC par le concepteur et d'en automatiser certaines tâches. A la manière de systèmes d'apprentissage et de révision interactive comme MIS ([Shapiro, 1981]), APT ([Nédellec, 1996]) ou MOBAL ([Morik, Wrobel, Kietz et Emde, 1993]), REVINOS est donc centré sur une coopération utilisateur-système en vue de corriger et/ou enrichir la BC.

Il parait légitime de se demander comment s'insère la révision coopérative dans la démarche d'acquisition des connaissances présentée dans la section 2.3.2. En première analyse, c'est au cours de l'étape "modéliser une autre situation concrète" (étape *e*) que REVINOS est utile au concepteur. Au cours de cette étape, le concepteur veut modifier un graphe déjà créé pour y intégrer une nouvelle situation concrète. L'"intégration d'une nouvelle situation concrète" n'est pas toujours un enrichissement de la BC : il s'agit souvent d'une *correction* dans laquelle le concepteur souhaite modifier la BC de manière à ce que dans la situation précisément identifiée, une action *b* soit effectuée à la place de l'action *a* jusqu'alors proposée par le système. C'est au sens large de *modification* (enrichissement et/ou

[1] REVINOS pour : outil de REVision Interactive de NOdules de Situation.

correction) que nous employons le terme *révision*. Dans ces conditions, on peut voir l'étape "modéliser une autre situation concrète" de la section 2.3.2 comme une tâche de révision.

L'outil de révision coopérative va prendre en main le concepteur pour guider le processus de révision depuis le moment où le concepteur a détecté une anomalie (i.e. une action non pertinente pour une situation) jusqu'à la validation des corrections proposées par le système. L'intérêt du recours à un outil de révision coopérative est double pour le concepteur :

1) Il permet bien sûr d'**automatiser certaines tâches** au cours du processus de révision. Nous montrerons comment REVINOS est capable de corriger les règles de choix automatiquement. REVINOS permet ainsi au concepteur d'être déchargé des problèmes "techniques" comme : "comment corriger les règles du nodule X pour que dans telle situation précise, le nodule Y soit activé ?". Le concepteur peut alors mieux se concentrer sur les problèmes "conceptuels" comme : "la situation concrète X et la situation concrète Y sont-elles suffisamment similaires pour être représentées par un même objet dans la BC ?"

2) Mais également l'outil de révision coopérative **guide** le concepteur de bout en bout durant le processus de révision :

- le dialogue que met en oeuvre REVINOS avec le concepteur est exactement adapté au problème courant de révision. Pour que la BC soit finalement corrigée de manière adéquate, le concepteur répond simplement aux questions du système : c'est REVINOS qui mène et oriente le dialogue tout au long de la phase de révision. Nous avons progressivement élaboré une expertise de révision des graphes de situation et celle-ci est utilisée par REVINOS pour faire en sorte que la tâche de révision soit autant que possible simple, facile et efficace pour le concepteur. L'identification de la cause de la révision et le choix de l'opérateur de révision sont par exemple faits par le système, à la lumière des informations données par l'utilisateur.

- REVINOS "gère" l'incrémentalité. Les nouvelles situations à corriger, mises à jour au cours de la correction de la situation courante, sont gardées en mémoire par le système. Les corrections effectuées par ailleurs sur la BC sont éventuellement répercutées sur ces situations. Le système propose en temps voulu à l'utilisateur de se pencher sur ces situations qui restent à traiter.

En plus de l'intérêt de décharger le concepteur de certaines opérations fastidieuses et de lui faciliter la tâche de révision en prenant en main le déroulement des opérations, l'outil de révision est capable d'offrir certaines fonctionnalités que le

concepteur ne serait pas en mesure d'effectuer aisément tout seul, alors même que ces fonctionnalités sont importantes dans le processus de révision :

3) REVINOS met précisément **en évidence les répercussions** des corrections effectuées. Le système sait calculer exactement les conséquences d'une correction en terme de couverture du graphe, notion que nous introduirons dans la section 3.3. Il est ainsi possible de valider la correction auprès de l'utilisateur en lui affichant toutes les situations modifiées par la correction effectuée. Mieux encore, inversement, pour choisir l'opérateur de correction adéquat, le système peut interroger l'utilisateur à propos des répercussions souhaitées. A cette fin REVINOS génère des exemples qu'il soumet à l'utilisateur.

4) REVINOS cherche à **généraliser la correction** effectuée à des situations similaires. En effet, le système peut éventuellement détecter et proposer des corrections similaires à celles qui ont été effectuées avec succès.

Ainsi, REVINOS cherche à répondre à nos objectifs de "révision située" que nous avons défini dans la section 1.2.3.2., puisque l'outil possède en résumé les fonctionnalités suivantes :

- il essaie de faire en sorte que la révision soit facile et locale[2],

- il stocke les cas de résolution insatisfaisants et permet à l'utilisateur de les traiter successivement,

- il cherche à valider la correction au moyen d'exemples concrets soumis à l'utilisateur,

- il cherche prudemment à généraliser la correction.

Ce chapitre décrit le déroulement général d'une session de révision avec REVINOS. Nous détaillerons précisément les trois grandes étapes de la coopération dans REVINOS (identification d'une cause d'erreur, correction, validation de la correction), avant de décrire une quatrième étape optionnelle, de généralisation de la correction. Tout au long de cette présentation du dialogue coopératif que propose REVINOS, nous exposerons les fonctionnalités propres de REVINOS en établissant des liens avec d'autres travaux et outils de révision coopérative et d'élicitation des connaissances.

[2] La révision est *locale* au sens où la correction effectuée concerne uniquement le cas qui a initié la révision : la correction n'est étendue aux autres cas qu'à la demande du concepteur.

3.1 Le déroulement général de la session de révision

La session de révision démarre pour une *situation courante* au sens donné dans la section 2.3.1. Rappelons pour mémoire qu'une situation courante est la conjonction d'un nodule, d'une trace et d'un état de la MT (cf. section 2.3.1). Dans ce chapitre, nous parlerons également d'*exemple courant* .

La boucle de révision dans REVINOS se décompose en trois étapes, comme le montre la figure 3.1.

tant qu'il y a un exemple à corriger
étape 1 : identification de la cause de l'erreur.
étape 2 : application d'un opérateur de révision, si le graphe nécessite une correction.
étape 3 : validation de la révision, par soumission des répercussions de la correction à l'utilisateur. Détection éventuelle de nouveaux exemples à corriger.
fintantque
étape 4 : généralisation éventuelle de la révision

Figure 3.1. Déroulement de la session de révision

D'une manière générale, au cours de chacune des étapes l'utilisateur est sollicité par le système pour effectuer certaines tâches. Le système peut par exemple au cours de l'étape 1 inciter l'utilisateur à créer ou modifier des objets de la BC. L'utilisateur se laisse toujours guider en répondant aux questions du système, éventuellement pour ce faire en “explorant pour comprendre” les objets de la BC que le système lui soumet. Dans la figure 3.2 par exemple, le système demande quelle action doit remplacer l'action “demander: conditionsFacCir”.

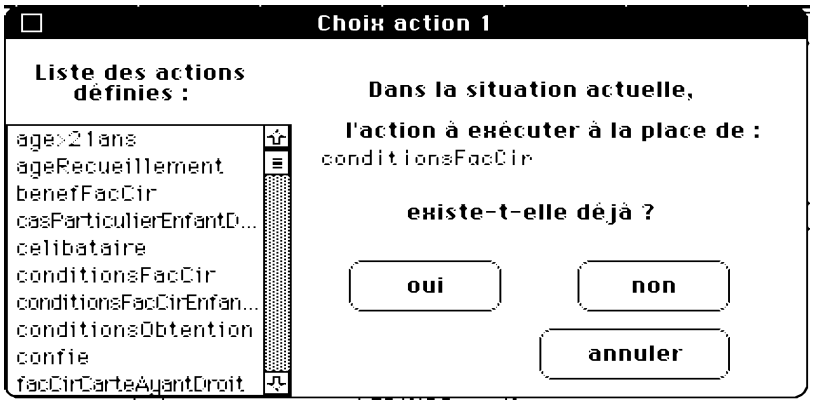


Figure 3.2 *Un extrait de dialogue durant l'étape 1*

Lors de l'étape 3 de validation, le système soumet à l'utilisateur les exemples qui correspondent aux différentes répercussions de la correction effectuée. En cas de non validation d'un exemple, le système repart à l'étape 1 avec un nouvel exemple, qui devient l'exemple courant. Cela ne signifie pas que la correction effectuée n'a pas été pertinente : l'enrichissement du graphe n'avait simplement pas été suffisant. Ainsi, dans le cas où une information manque dans la situation courante pour pouvoir décider de l'action pertinente, une première correction modifiera le graphe de sorte que, dans le même contexte, l'information soit désormais connue grâce à l'exécution d'une action *a*. Mais ni les actions qu'il convenait de faire dans cette situation, ni celles qu'il faut effectuer dans les situations où *a* renvoie d'autres informations, n'ont été précisées : d'autres corrections, donc d'autres "passes" dans la boucle de révision de la figure 3.1, seront nécessaires. Par convention, nous dirons que chaque passage dans la boucle de l'algorithme de la figure 3.1 est une *révision* : une session de révision est donc constituée de plusieurs révisions successives.

Après validation de tous les exemples, la session de révision peut se poursuivre éventuellement si le système détecte une généralisation possible des corrections apportées (étape 4).

Nous allons maintenant détailler chacune de ces quatre étapes.

3.2 Etape 1 : identification de la cause de l'erreur

L'étape 1 consiste à identifier la cause du désaccord entre le système et le concepteur. Cette étape 1 fait déjà un grand pas vers l'identification de la réparation associée à "l'erreur". Le dialogue dont la figure 3.2 est un extrait, permet à REVINOS d'aboutir à l'identification d'une "cause d'erreur".

Trois grandes causes d'erreur peuvent être distinguées, qui correspondent à trois types de réparations :

a) Situation inédite : les informations de la MT reflètent de manière correcte et complète la *situation concrète* que se représente l'utilisateur (voir section 2.3.1), mais l'action proposée n'est pas pertinente. L'action pertinente et le nodule correspondant à cette situation peuvent être spécifiquement créés, ou réutilisés si des objets déjà existants conviennent. Cette cause d'erreur est la plus courante et aboutit à une *correction du graphe* au sens où pour la résoudre il est nécessaire que des règles de choix soient modifiées.

b) Action à corriger : les informations de la MT reflètent de manière correcte et complète la situation concrète mais l'action proposée n'est **jamais** pertinente pour

le nodule : pour tous les autres cas liés au nodule, la nouvelle action est pertinente. Il convient alors de simplement changer l'action du nodule courant.

c) Procédure à corriger : une information de la MT n'est pas correcte. Il faut corriger la procédure qui a renvoyé cette information erronée.

Les causes b) et c) ne nécessitent pas la correction des règles d'un nodule : après la correction de l'action (simple correction du champ “actions” d'un nodule ou véritable correction d'une procédure ou d'un écran), le système passe directement à l'étape 3 de validation.

En revanche, la cause a) nécessite ou bien l'accrochage d'un nouveau nodule au graphe, ou bien la correction d'une “erreur d'aiguillage” : c'est ce qui est fait durant l'étape 2, de correction proprement dite. Cette étape 2 commence par l'identification d'un *objectif de correction*, c'est-à-dire la conjonction d'un nom de nodule courant et d'un nom de “nodule cible”, à atteindre à partir du nodule courant. Puis l'application d'un opérateur de révision fera en sorte de modifier les règles du nodule courant (appelé <nodDépart> par la suite) afin qu'au moins pour l'exemple courant, noté <exCourant>, le nodule cible, noté <nodCible>, soit activé.

3.2.1 Le graphe “REVINOS”

Le système REVINOS se charge d'identifier la cause de l'erreur parmi les trois catégories que nous avons distinguées dans la section précédente. Pour cela il pose quelques questions à l'utilisateur. Ce dialogue a été défini sous la forme... d'un graphe de nodules de situation. En effet, il s'agit là d'un problème de classification, problème pour lequel la représentation par graphe de nodules est parfaitement adaptée[3], surtout lorsque les informations nécessaires à la classification doivent être extraites progressivement (en l'occurrence le système va chercher ces informations auprès de l'utilisateur). Ce graphe, appelé tout simplement “REVINOS” apparaît dans la figure 3.3.

[3] D'autant plus que ce graphe a longuement été lui-même révisé et que nous avons conçu notre représentation pour faciliter la révision. L'application n'a toutefois jamais été prévue pour fonctionner de manière réflexive (nous n'avons pas cherché à permettre l'application du graphe “REVINOS” à lui-même)...

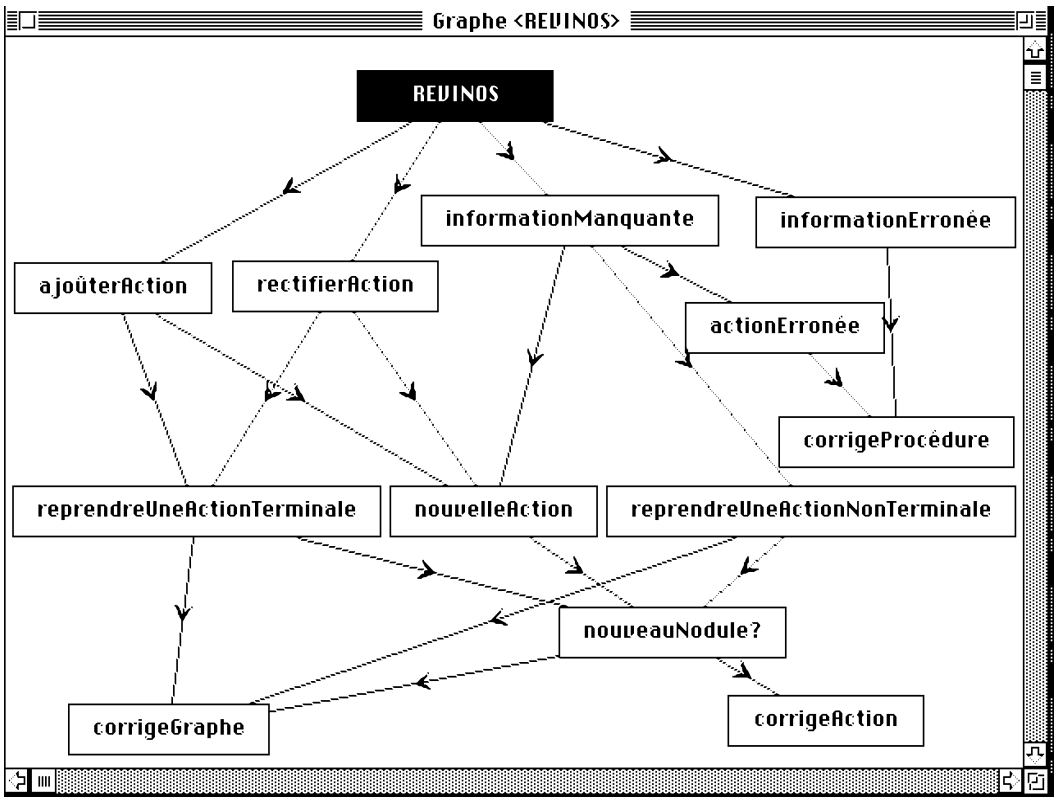


Figure 3.3 : Le graphe “REVINOS”

Comme le graphe “REVINOS” le fait apparaître, plusieurs parcours différents mènent à une même cause d'erreur. Les trois causes “situation inédite”, “action à corriger” et “procédure à corriger” correspondent respectivement aux trois nodules “corrigeGraphe”, “corrigeAction” et “corrigeProcédure”. Le nodule “corrigeGraphe” appelle un sous-graphe, lui-même nommé “corrigerGraphe” conformément aux conventions de dénomination des nodules définies dans la section 2.3.3.

La première question posée à l'utilisateur (voir figure 3.4) est déterminante et va orienter le dialogue de manière décisive. Elle suppose un examen approfondi de la MT par l'utilisateur. En plus des informations que donnaient déjà la fenêtre “Visualisation de la situation courante” d'EDINOS (voir figure 2.8), l'utilisateur a la possibilité de démarrer le processus de révision en choisissant parmi trois “premiers diagnostics” possibles. En lançant le processus de révision, l'utilisateur exprime son désaccord quant à l'action à effectuer dans cette situation. Il doit choisir parmi trois débuts d'explication possibles :

- les informations contenues par la MT sont vérifiées dans la situation concrète que se représente l'utilisateur et elles suffisent pour décider de l'action à effectuer (qui n'est malheureusement pas celle proposée par le système),

- il manque une information pour être véritablement en mesure de décider de l'action à effectuer,

- une information de la MT est discordante d'avec la situation concrète, donc on peut considérer que l'information en question est erronée.

Nous allons maintenant passer en revue chacun de ces trois scénarios.

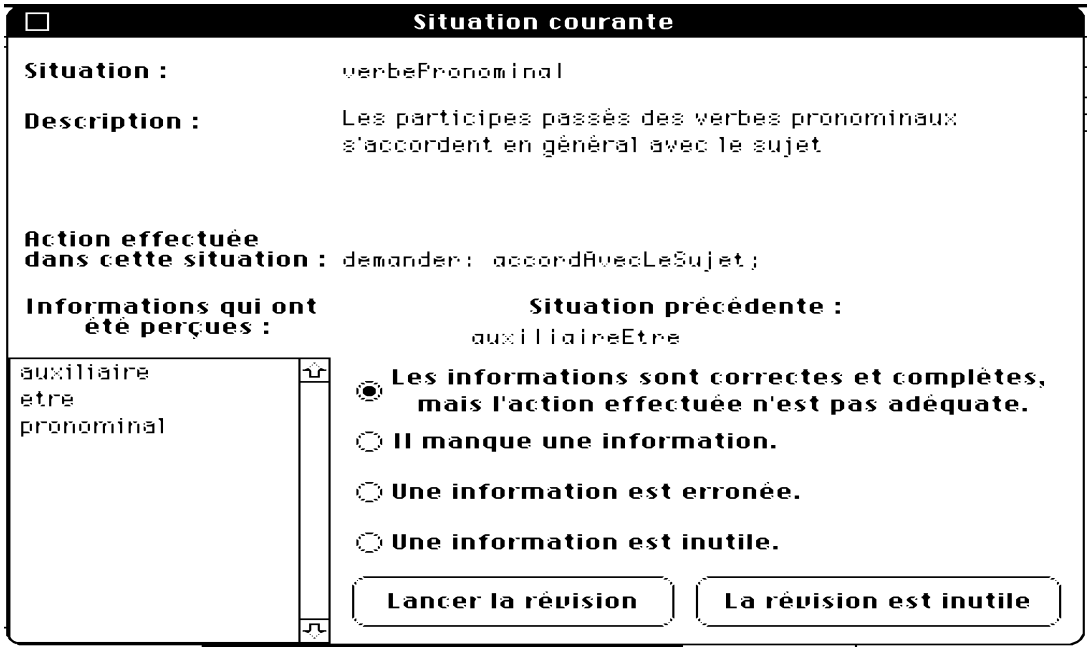


Figure 3.4 : Question posée depuis le nodule “REVINOS” de la figure 3.3

3.2.1.1 Cas où les informations de la MT sont correctes et complètes

Dans le cas où les informations de la MT correspondent exactement à la situation concrète, le chemin parcouru pour arriver au nodule courant est presque intégralement correct : seul le choix du nodule terminal doit éventuellement être corrigé. La question principale qu'il reste à régler est la suivante : est-ce que l'action courante “erronée”, puisque rejetée par l'utilisateur, l'est aussi pour toutes les autres situations courantes du nodule ? Rappelons en effet qu'un nodule peut être lié à plusieurs situations courantes. La réponse à cette question permettrait en effet de choisir entre les deux causes / réparations “situation inédite” et “action à

corriger". Nous verrons que nous avons pour l'instant fait le choix de ne pas poser explicitement cette question à l'utilisateur : à un moment de la session de révision, l'utilisateur choisit simplement de créer un nouveau nodule ou non. L'idée importante sur laquelle nous voudrions insister à cette occasion est la suivante : avant que toutes les répercussions des corrections ne soient validées par l'utilisateur, les choix qui ont pu être effectués durant la phase de révision par le système ou par l'utilisateur, de manière consciente ou non, ne sont jamais définitifs. D'une manière générale tout au long du processus de révision, nous avons toujours l'alternative suivante, en tant que concepteur de l'outil de révision :

- ou bien le choix à faire est effectué par le système de manière autoritaire au moyen d'heuristiques, la validation du choix sera alors faite au moment de la validation des répercussions des corrections;

- ou bien l'on tente au mieux d'anticiper et d'exposer à l'utilisateur les conséquences du choix afin d'éviter de différer la validation de ce choix.

Notre objectif de construire un système de révision *coopérative* nous fait préférer la seconde option. Or, celle-ci ne peut pas toujours facilement être mise en oeuvre parce qu'elle nécessite de pouvoir expliciter immédiatement et simplement les conséquences du choix afin de les présenter à l'utilisateur. La première option est la plus simple et la plus économique lorsque différer le choix dans le temps n'est pas coûteux. Ainsi, dans le cas présent, il sera toujours temps de se pencher sur les "exemples voisins" (les autres situations courantes liées au même nodule) lors de la phase de validation des répercussions et cela ne portera pas à conséquence pour l'exemple courant . Nous avons donc retenu la première option.

Revenons au graphe "REVINOS" et examinons les branches qui correspondent à ces cas où les informations de la MT sont correctes et complètes. L'étape suivante après la question de la figure 3.4, consiste à repérer ou créer de toute pièce l'action adéquate pour la situation courante. La figure 3.2 montrait la question posée depuis le nodule "rectifierAction" : "L'action à exécuter à la place de ... existe-t-elle déjà ?". Or la formulation de la question ne doit pas être exactement celle-ci dans tous les cas. En effet, la session de révision démarre en général à partir d'une *interruption* de simulation. Mais elle peut aussi démarrer en *fin* de simulation : lorsque le nodule courant est terminal. La différence essentielle entre une "simulation interrompue" et une "simulation terminée", c'est que dans le second cas l'action du nodule courant a déjà renvoyé un fait et que celui-ci n'a pas suffi pour choisir un nouveau nodule. Dans la question qu'il convient alors de formuler, il ne s'agit pas de remplacer la dernière action exécutée, mais de simplement identifier l'action à exécuter dans la situation courante. C'est la question posée depuis le nodule "ajouterAction". Pour choisir entre "ajouterAction" ou

"rectifierAction", REVINOS contrôle simplement si l'action courante a déjà renvoyé un fait ou non.

Si les deux nodules "ajouterAction" et "rectifierAction" contiennent deux questions formulées légèrement différemment, ils se comportent toutefois de la même manière et activent tous les deux l'un des deux nodules suivants :

- le nodule "nouvelleAction" dans lequel l'utilisateur est incité à créer la nouvelle action qu'il désire (voir figure 3.5). L'utilisateur / concepteur doit alors donner un nom à la nouvelle procédure ou au nouvel écran, définir le contenu du nouvel écran (pour le cas d'une action de type "demander"), et dans tous les cas, remplir le champ "description" définissant les faits que peut renvoyer l'action.

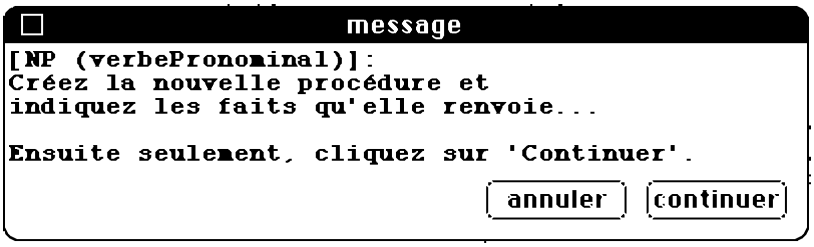


Figure 3.5 : Action exécutée depuis le nodule "nouvelleAction"

- le nodule "reprendreUneActionTerminale" correspond à la situation où l'utilisateur a choisi de reprendre ou réutiliser une action déjà créée. REVINOS lui propose alors de manière un peu abrupte de reprendre éventuellement un des nodules déjà créés qui effectue l'action en question (voir figure 3.6).

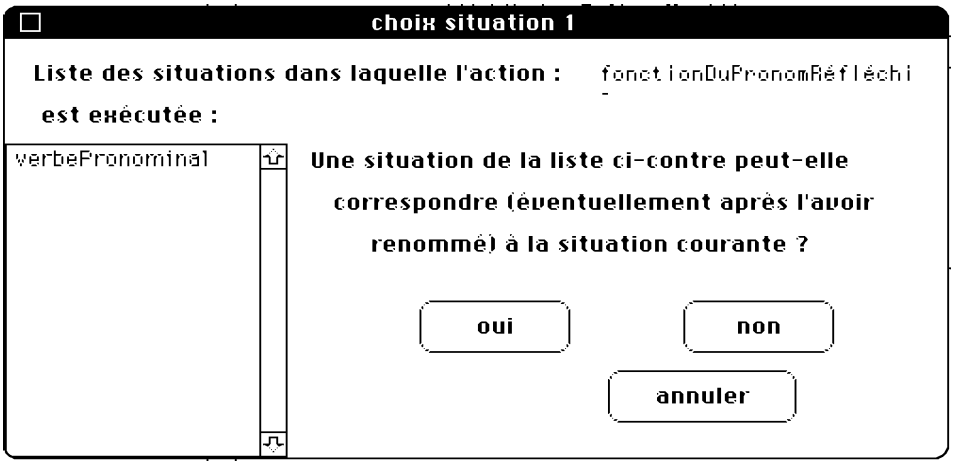


Figure 3.6 : Question posée depuis le nodule "reprendreUneActionTerminale"

Si l'utilisateur répond négativement à la question de la figure 3.6 ou dans le cas où il a créé une nouvelle action, REVINOS lui demande de donner un nom au nouveau nodule qui va être créé (figure 3.7). La liste de nodules déjà créés est affichée pour aider l'utilisateur à trouver un nom “cohérent” avec le reste de la BC, c'est-à-dire respectant les éventuelles conventions dans la dénomination des nodules. L'utilisateur peut visualiser le contenu de ces autres nodules pour mieux les comprendre s'il le désire, et rechercher si l'un d'entre eux ne correspondrait pas exactement à la situation concrète actuelle.

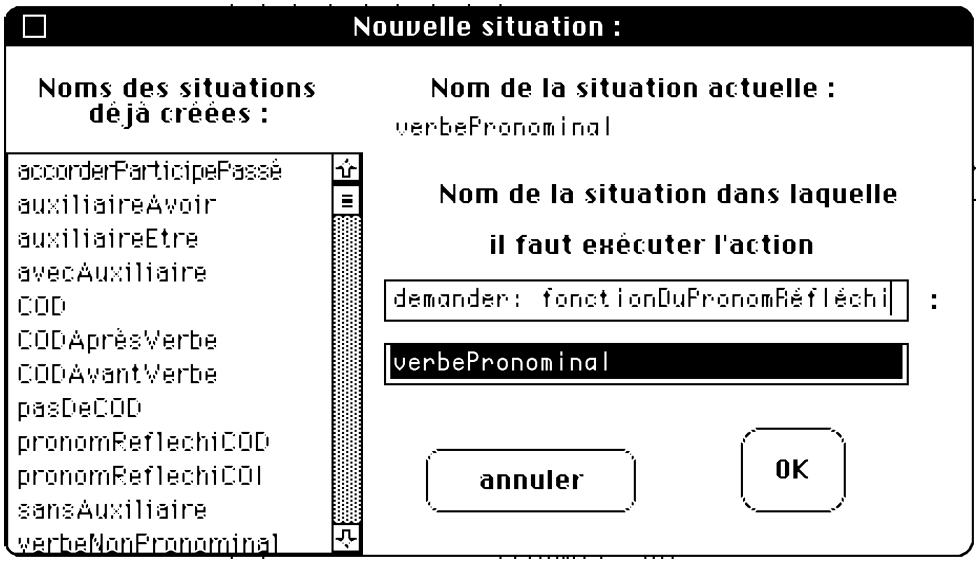


Figure 3.7 : Question posée depuis le nodule “nouveauNodule?”

Par défaut, REVINOS propose comme nom du nouveau nodule, le nom du nodule courant : c'est à ce point précis du dialogue que l'utilisateur peut choisir de conserver le même nom pour le nouveau nodule, c'est-à-dire de remplacer l'ancien nodule. Si c'est le choix fait par l'utilisateur, REVINOS activera ensuite le nodule “corrigeAction”, qui poursuit la session de révision par l'étape de validation (la correction effectuée consiste à simplement changer l'action du nodule courant). Par ce “remplacement” du nodule courant, nous considérons que l'utilisateur sait qu'il effectue une action qui a plus de conséquences que le “simple” ajout d'un nouveau nodule pour caractériser la situation courante comme cas particulier. Une amélioration du dialogue consiste à insister davantage sur ce choix et faire apparaître clairement les “exemples voisins” concernés par le renommage du nodule courant. Nous avons déjà présenté au début de cette section cette alternative qui consisterait à donner à l'utilisateur tous les éléments pour qu'il fasse son choix délibérément. Nous avons préféré retenir la solution la plus simple qui consiste à ne pas insister

sur le choix en question, quitte même à ce que ce choix soit effectué presque à l'insu de l'utilisateur. Nous avons en effet cherché d'abord à construire un dialogue de révision pour un utilisateur novice, conformément à notre objectif général de rendre la révision aussi facile que possible. L'étape suivante dans l'amélioration du dialogue de REVINOS est de permettre à un utilisateur plus “expert” de paramétrer le déroulement de la session de révision et de pouvoir définir au départ explicitement ses préférences sur la manière dont ce type de scénario doit se dérouler.

3.2.1.2 Cas d'une information manquante

Depuis le nodule “informationManquante”, REVINOS incite l'utilisateur à créer, reprendre ou corriger l'action qui peut renvoyer cette information qui fait défaut (voir figure 3.8). Le “cas d'une information manquante” pourrait la plupart du temps être qualifiée de “cas d'une action manquante” : il s'agit en effet d'identifier cette action qu'il convient d'intégrer au graphe courant. On se ramène alors à un problème similaire au cas où les informations de la MT sont correctes et complètes.

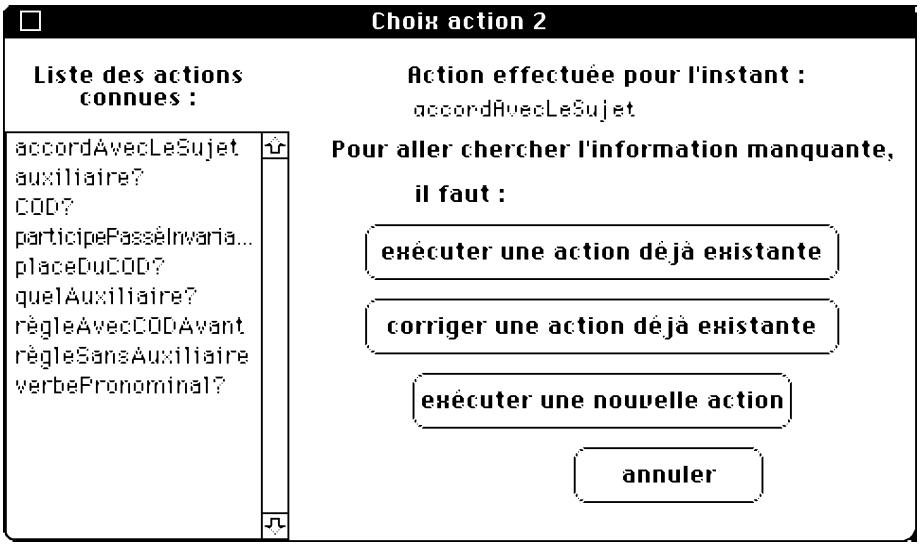


Figure 3.8 : Question posée depuis le nodule “informationManquante”

Les trois cas de figure sont les suivants :

- Il se peut que l'action qui aurait renvoyé le fait manquant si elle avait été exécutée durant le parcours de l'exemple courant, existe déjà (“exécuter une action déjà existante”).

- Il se peut également qu'une action existe déjà mais ne convienne pas tout à fait (“corriger une action déjà existante”). Ainsi, la correction d'une action peut aboutir à l'ajout du fait manquant dans la liste des faits renvoyés par l'action.

- Enfin, l'utilisateur peut être obligé de créer une nouvelle action qui renverra le fait manquant. On retrouve alors le nodule “nouvelleAction” présenté dans la section précédente.

Le cas d'une action reprise ou réutilisée correspond également à une situation déjà vue dans la section précédente : le nodule “reprendreUneActionTerminale”. Mais de même qu'entre les nodules “ajouterAction” et “rectifierAction”, la différence porte là aussi sur la formulation de la question. Le nodule “reprendreUneActionNonTerminale”, qui correspond au cas d'une information manquante, propose la liste des nodules exécutant l'action sélectionnée auparavant. Or la question ne peut pas être formulée de la même manière que dans la figure 3.6 car l'action choisie n'est pas terminale puisqu'elle doit servir à collecter l'information manquante nécessaire préalablement au choix du nodule final. Le nodule qui sera éventuellement réutilisé ne sera donc pas terminal, mais un simple point de passage. La figure 3.9 montre la question formulée dans ce contexte.

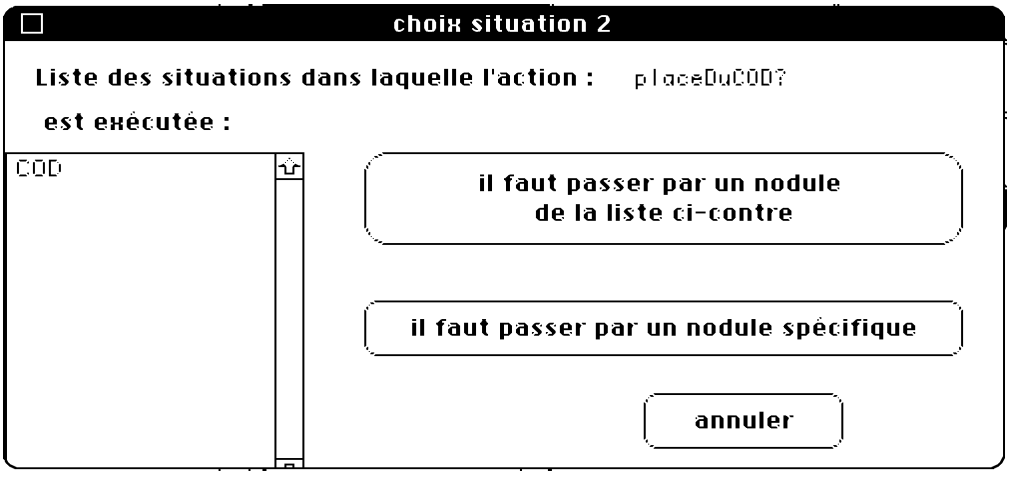


Figure 3.9 : Question posée depuis le nodule “reprendreUneActionNonTerminale”

Il faut noter que le cas d'une information manquante, à la différence du cas où les informations de la MT sont correctes et complètes, nécessite toujours au moins deux révisions au sens donné dans la section 3.1. Si la première révision a pour conséquence d'avoir pour la situation courante des informations correctes et complètes dans la MT, la seconde révision est nécessaire pour préciser l'action finale adéquate.

Le cas d'une information erronée est similaire en ce sens qu'il nécessite également au moins deux révisions : une première pour corriger l'action en cause, une seconde pour préciser l'action finale adéquate.

3.2.1.3 Cas d'une information erronée

Le dernier cas de figure est le plus simple. Dans le cas d'une information erronée, l'utilisateur doit sélectionner le fait en cause dans la fenêtre de la figure 3.4. Puis le nodule "informationErronée" l'invite à corriger l'action qui a renvoyé ce fait (voir figure 3.10), de la même manière que dans la figure 3.5 l'utilisateur était invité à créer une nouvelle procédure.

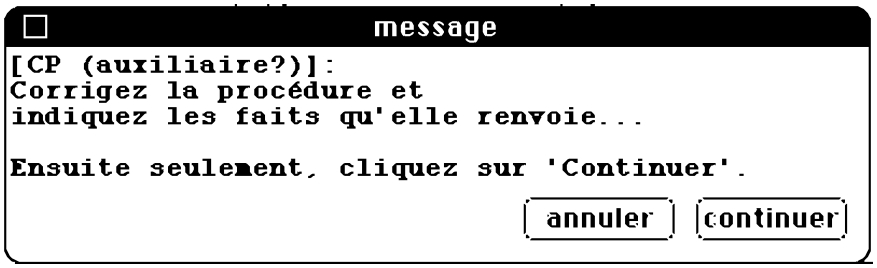


Figure 3.10 : Action exécutée depuis les nodules "informationErronée" et "actionErronée"

L'action du nodule "informationErronée" est la même que celle du nodule "actionErronée" pour le cas d'une information manquante : ce sont là les deux cas de figure qui correspondent à la cause c) "procédure à corriger" présentée dans la section 3.2. Dans les deux cas, le nodule "corrigeProcédure" est ensuite activé. Celui-ci déclenche le passage à l'étape 3 de validation de la correction effectuée. Rappelons que pour ces deux cas, la correction a été effectuée uniquement dans la Base d'Actions (écrans ou procédures).

Que les deux cas de figure correspondant aux deux nodules "informationErronée" et "actionErronée" soient très similaires n'est pas étonnant

parce que ces deux parcours possibles dans le graphe REVINOS correspondent à deux interprétations différentes d'un même problème : il est heureux dans ces conditions que la solution proposée par REVINOS soit la même dans les deux cas.

Pour illustrer cela, prenons une situation concrète pour laquelle le graphe courant propose à un moment donné une action qu'on pourrait qualifier "d'incomplète", par exemple une question posée qui ne propose pas parmi l'ensemble des réponses possibles de réponse convenable pour la situation concrète :

- ou bien l'utilisateur interrompt immédiatement la simulation, c'est-à-dire dès l'apparition de cette question. Il choisira alors de cocher "une information manque" dans la fenêtre de la figure 3.4.

- ou bien l'utilisateur choisit la réponse qui se rapproche le plus de la situation concrète et poursuit le dialogue jusqu'à ce qu'il aboutisse à un désaccord plus important. Il choisira alors de cocher "une information est erronée" lors du premier diagnostic. Il se peut même qu'en fin de compte l'utilisateur trouve le parcours et l'action finale corrects, au seul détail près de la présence de ce fait "douteux" : la révision peut alors être déclenchée dans le seul but d'enrichir l'action "imparfaite", de telle sorte qu'elle puisse renvoyer un fait qui corresponde plus précisément à la situation actuelle. REVINOS prend parfaitement en charge ce type de session de révision triviale.

3.2.1.4 Un exemple

Pour illustrer le déroulement général d'une session de révision avec REVINOS et pour illustrer concrètement une utilisation du graphe "REVINOS", nous allons détailler une révision assez simple d'un graphe ayant pour but d'aider à bien accorder les participes passés. L'accord du participe passé en français est un problème de classification qui peut facilement se modéliser progressivement sous la forme de graphes de nodules de situation. Le grand intérêt de ce domaine est que chaque cas résolu est une phrase en français, par exemple "Ils se sont succédé" ou "Elle s'est lavée". Il suffit donc d'énoncer la phrase pour décrire parfaitement la situation concrète qu'on voudrait intégrer à la BC...

La figure 3.11 affiche le graphe "accorderParticipePassé" avant la révision.

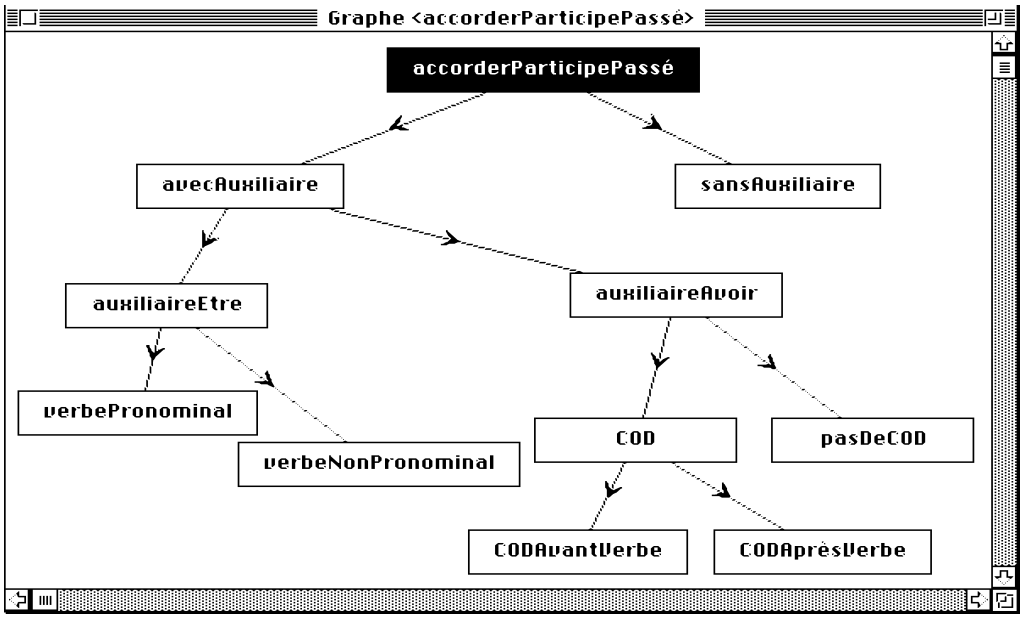


Figure 3.11: Le graphe “accorderParticipePassé”

Le graphe est exécuté pour l'exemple “Ils se sont succédé”. L'utilisateur répond à trois questions avant que le système ne donne sa conclusion : “Le participe passé s'accorde en genre et en nombre avec le sujet” (écran “accordAvecLeSujet”). L'utilisateur / concepteur décide de visualiser la situation courante et REVINOS ouvre alors la fenêtre de la figure 3.4. A cet instant, le nodule final est “verbePronominal”. Les trois faits de la MT signifient que la phrase contient un auxiliaire, que cet auxiliaire est le verbe être, et que le verbe utilisé (se succéder) est pronominal.

Le concepteur veut corriger le graphe et préciser que le participe passé ne doit ici pas s'accorder parce que le pronom réfléchi assure la fonction de Complément d'Objet Indirect (COI)[4].

A la question de la figure 3.4, le concepteur répond qu'il manque une information (la fonction du pronom réfléchi) et lance la session de révision. Après la question de la figure 3.8, l'utilisateur va créer l'action “fonctionDuPronomRéfléchi?” (figure 3.5). Il va par exemple formuler la question de la manière suivante : “Le pronom réfléchi est-il COD ou COI du verbe ?”. En

[4] Rappelons que cette session de révision est un extrait de la construction incrémentale du graphe “accorderParticipePassé” : le graphe ne sera pas encore valide à la suite de cette correction. Plus précisément, il convient en fait de prendre en compte la fonction du pronom réfléchi uniquement lorsqu'il s'agit d'un verbe pronominal réciproque (c'est le cas de “se succéder") ou réfléchi. L'exemple “elle s'est enfuie” amènera ensuite le concepteur à faire la distinction entre les quatre types de verbes pronominaux : verbes pronominaux réciproques, verbes pronominaux réfléchis, verbes essentiellement pronominaux (“elle s'est enfuie”) et verbes pronominaux de sens passif (“les petits pains se sont bien vendus”). Cette distinction sera en fin de compte faite dès le nodule “verbePronominal" de la figure 3.11.

fonction de la réponse de l'utilisateur, l'action renverra le fait “pronomReflechiCOD” ou “pronomReflechiCOI”.

Nous arrivons alors au nodule “nouveauNodule?” du graphe REVINOS (figure 3.3), après avoir parcouru les nodules “ajouterAction” et “nouvelleAction”. La question posée au concepteur est celle de la figure 3.7. Le concepteur choisit de conserver le nom “verbePronominal” pour le nodule courant, autrement dit de remplacer l'action “demander: accordAvecLeSujet” par “demander: fonctionDuPronomRéfléchi?” dans ce nodule.

Nous aboutissons finalement au type de réparation “action à corriger” si l'on suit la typologie de la section 3.2 (nodule "corrigeAction” du graphe REVINOS).

REVINOS passe alors directement à l'étape 3 de calcul des répercussions. Il trouve logiquement deux exemples nouveaux : celui correspondant au cas où le pronom réfléchi est COD (qu'on appellera “exempleCOD” par la suite) et celui correspondant au cas où le pronom réfléchi est COI (notre exemple initial : “Ils se sont succédé”). Il en choisit un, “exempleCOD”, qu'il soumet à l'utilisateur (figure 3.12). Comme il n'y a pas de règles de choix pour le nodule “verbePronominal”, l'action proposé pour cette exemple est vide. Il est logique que l'utilisateur refuse de valider cet exemple et qu'une nouvelle révision soit lancée pour “exempleCOD” (figure 3.13). Ensuite nous reviendrons à notre exemple initial (“ils se sont succédé”), pour lequel l'action “demander: fonctionDuPronomRéfléchi?” renvoie “exempleCOI”. Une troisième session de révision sera lancée au cours de laquelle le concepteur indiquera qu'il convient d'exécuter l'action “participePasséInvariable” pour cet exemple.

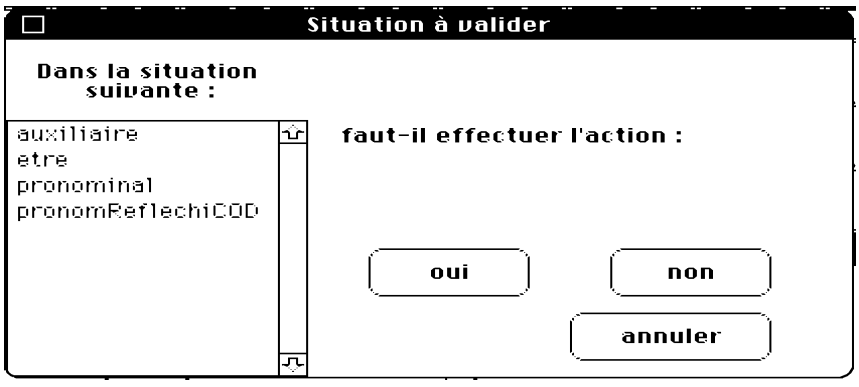


Figure 3.12 : Soumission d'un exemple au cours de l'étape 3

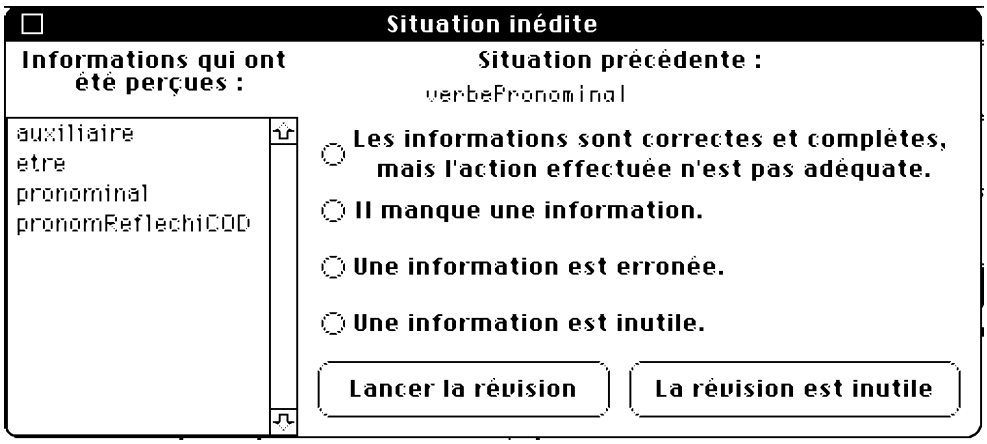


Figure 3.13 : "exempleCOD" nécessite une seconde révision

3.2.2 Travaux similaires

3.2.2.1 A propos du démarrage de l'étape de révision

[Nédellec, 1998] distingue trois grandes étapes dans une session de révision :

1) identification d'une erreur dans la BC,

2) identification des causes possibles de l'erreur (diagnostic),

3) sélection des causes et correction des erreurs correspondantes (réparation).

Si le point 2) correspond à notre étape 1 et si le point 3) correspond à nos étapes 2 et 3, le point 1) par contre n'apparaît pas dans la description du déroulement général de la session de révision dans REVINOS (figure 3.1). En effet, de même que la très grande majorité des systèmes apprentis, REVINOS laisse la tâche d'identification d'une erreur à l'utilisateur. L'utilisateur détecte les dysfonctionnements du système à partir de l'outil de simulation. Nous faisons l'hypothèse que l'utilisateur interrompt la simulation aussitôt qu'il se met à douter de la pertinence de l'action courante. Nous considérons donc qu'au moment de l'interruption, le parcours jusqu'au nodule père du nodule courant est valide du point de vue de l'utilisateur. Ce dernier a la possibilité de remonter dans le parcours ou de recommencer une simulation pour se positionner sur la première action litigieuse.

Par opposition, le système ODYSSEUS ([Wilkins, 1990]) est un système apprenti qui est capable de détecter une erreur dans la BC en cas "d'échec d'explication". ODYSSEUS est construit au dessus de NEOMYCIN, qui est une

version enrichie de MYCIN. Le système observe les actions d'un expert-médecin et, pour chaque action de l'expert, cherche à construire une explication, c'est-à-dire un lien entre l'action et le but général. Si ODYSSEUS ne parvient pas à construire une explication, il s'agit là d'un symptôme d'incomplétude de la BC.

"Les systèmes apprentis peuvent être définis comme des systèmes capables de corriger leur BC en observant un expert accomplissant la tâche. Un système apprenti doit non seulement savoir comment l'expert atteint sa décision finale, mais il doit aussi comprendre chaque action intermédiaire de résolution de problème. Lorsque l'apprenti ne comprend pas l'action effectuée par l'expert, il prend là l'occasion pour apprendre"

[Donoho et Wilkins, 1994, p. 14-1]

La vision de David Wilkins et de Steven Donoho de la notion de système apprenti est particulièrement exigeante puisqu'elle impose au système de surveiller et de comprendre le comportement d'un expert. Les systèmes apprentis sont plus couramment vus comme fonctionnant exactement de manière inverse (voir section 1.2.2.2) : l'expert doit surveiller le comportement du système et sa tâche est donc plus contraignante que celle de l'expert d'ODYSSEUS. Mais la tâche de compréhension du comportement d'autrui nous paraît trop difficile pour être pouvoir être effectuée par le système. Nous la laissons à l'utilisateur, tout en cherchant à lui donner le plus de moyens possible pour la remplir. [Donoho et Wilkins, 1994] soulèvent des problèmes importants auxquels ODYSSEUS2 est confronté à cause de l'objectif particulier qu'ils se sont fixé. Le problème du "masquage" par exemple est le problème de la construction de "mauvaises" explications qui empêchent le déclenchement d'un apprentissage pourtant nécessaire. Autrement dit, la BC permettant de construire des explications doit être "suffisamment correcte" pour que l'apprentissage puisse avoir lieu et que la BC en question puisse être améliorée...

3.2.2.2 A propos du déroulement général de la session de révision

a) Comparaison avec le système APT

Le système APT ([Nédellec et Causse, 1992], [Nédellec, 1996], [Nédellec, 1998]) apprend et corrige des règles de résolution de problème et pour cela utilise deux sources de connaissances : une théorie du domaine et un utilisateur, considéré comme expert du domaine.

La différence fondamentale avec REVINOS est la présence d'une théorie du domaine qui comprend notamment des hiérarchies de concepts. Du point de vue de l'apprentissage, cette théorie du domaine est utile :

1) au moment de la phase de complétion, qui peut être considérée comme l'équivalent de notre étape 1 d'identification de la cause de l'erreur. Cette étape consiste à trouver une "explication" de l'exemple courant en "complétant" ce dernier par des informations vérifiées dans la situation courante, mais qui n'apparaissent pas dans la description de l'exemple dans le langage du système. Le système DISCIPLE ([Tecuci et Kodratoff, 1990]) fonctionne de cette manière. Il s'agit donc de l'équivalent de la recherche des informations manquantes ou erronées dans la MT de REVINOS. Mais tandis que dans REVINOS cette sous-tâche de l'étape 1 est laissée complètement à la charge de l'utilisateur, APT se sert de sa théorie du domaine pour fabriquer des propositions d'explication et les soumettre à l'utilisateur.

2) au moment de la phase de généralisation de l'explication, APT utilise une méthode de "généralisation à petits pas" ([Nédellec, 1991]) qui se sert de la théorie du domaine, pour construire et soumettre à l'utilisateur des exemples discriminants (nous reviendrons sur ce point dans la section 3.6.2).

En revanche, l'inconvénient du recours à une théorie du domaine est que cette dernière peut être incorrecte ou incomplète. Cela peut mener APT à surgénéraliser la règle apprise et nécessiter une troisième phase, après la complétion et la généralisation, de révision (de la théorie du domaine et de la règle).

Comment comparer l'architecture d'APT avec celle de REVINOS ? Nous avons vu dans la section 2.1.3.2.c qu'une théorie du domaine pouvait très bien être disséminée dans la Base de Nodules, ou exister en tant que telle dans des Bases de Règles appelées par des action de type "déduire". Dans le premier cas, la théorie du domaine n'est pas structurellement distinguée du reste de la Base de Nodules, et la révision se déroule normalement, c'est-à-dire de la manière décrite dans la figure 3.1. Dans le second cas, la révision de la théorie du domaine a lieu au cours de notre étape 1, et prend la forme d'une correction d'action de type "déduire". Autrement dit, contrairement à APT, la révision de la théorie du domaine aurait lieu préalablement à l'apprentissage d'une nouvelle situation de résolution de problème.

b) Sur la notion d'incrémentalité

La boucle de révision de la figure 3.1 semble être une particularité de REVINOS par rapport aux autres systèmes d'apprentissage et de révision. L'originalité vient de ce que REVINOS est capable de générer des nouveaux cas de

figure que schématiquement l'utilisateur doit classer dans l'une des deux catégories suivantes : “correct” ou “à corriger”. Tandis que les algorithmes d'apprentissage ou de révision qui classiquement cherchent à apprendre la définition d'un concept, demandent à l'utilisateur de ranger les exemples dans les catégories “vrai” ou “faux”, voir “impossible”.

Il y a deux explications à cette différence. D'abord REVINOS est construit sur une approche complètement incrémentale (voir section 1.1.2 et 2.6) : son but est prioritairement d'**intégrer l'exemple courant** à la BC, non de chercher à apprendre ce qu'on pourrait appeler “une définition complète du nodule courant”, en recensant l'ensemble des situations possibles appartenant à cette classe de situations dans lesquelles il convient d'effectuer l'action *a*. C'est là une propriété des algorithmes d'apprentissage incrémental que d'être focalisés sur l'exemple courant à intégrer, et non sur une “classe” à apprendre.

Mais la deuxième particularité de REVINOS, c'est de faire partie d'un processus de “modélisation opérationnelle incrémentale” (section 2.6.2). La méthodologie d'acquisition des connaissances basée sur notre modèle de représentation des connaissances mettait déjà l'accent sur l'importance de détecter de nouvelles situations concrètes à modéliser, au cours de la modélisation d'une première situation concrète initiale (section 2.3.2). REVINOS permet d'en découvrir automatiquement en faisant le bilan des répercussions de la correction effectuée, comme nous le montrerons dans la section 3.4. Il est dans ce contexte logique de ne jamais se trouver en présence de véritables exemples qu'on appelle “négatifs” dans la terminologie de l'apprentissage automatique (parce qu'il ne faut jamais les couvrir), mais toujours “d'exemples à intégrer”. Et REVINOS se doit de mémoriser et mettre à jour cet ensemble d'exemples détectés qu'il faudra tôt ou tard “apprendre”[5].

Finalement, REVINOS fait de l'apprentissage multi-classes : l'exemple à corriger peut être vu temporairement comme un exemple négatif pour un nodule donné, mais le concepteur doit très vite spécifier ou créer le nodule adéquat pour cet exemple. L'ancien exemple négatif devient tout de suite exemple positif d'un deuxième nodule.

[5] La question de la convergence globale de l'apprentissage est difficilement abordable car elle repose entièrement entre les mains du concepteur, qui doit être méthodique et rigoureux. De manière générale, on peut douter de la pertinence de la question de la convergence d'une méthode d'acquisition des connaissances.

3.2.2.3 Le dialogue avec l'utilisateur durant le diagnostic

Le dialogue de REVINOS au cours de l'étape 1 partage quelques points communs avec les dialogues des systèmes TEIRESIAS, ASK et ETS, qui sont tous trois des outils interactifs d'acquisition des connaissances.

a) TEIRESIAS

REVINOS pourrait se réclamer de la descendance du système TEIRESIAS ([Davis, 1979]), qui est un assistant intelligent pour la construction et la maintenance de BC. Randall Davis insiste sur l'importance d'acquérir les connaissances en contexte. Par conséquent, de manière identique à REVINOS, l'utilisateur, vu comme un tuteur du système, peut interrompre le système à tout moment au cours de son fonctionnement et lancer une phase de débogage (principe des systèmes apprentis).

Les trois premières questions que pose TEIRESIAS à l'utilisateur sont les suivantes :

- Tous ces résultats sont-ils corrects ?
- En manque-t-il ?
- Y en a t-il qui n'auraient pas du apparaître ?

La similitude avec la première question de REVINOS (figure 3.4) est flagrante. [Davis, 1979] développe en détail un exemple de session de débogage. L'utilisateur et le système passent en revue les règles invoquées et arrivent à la conclusion que toutes les règles sont correctes, mais qu'il en manque une qui puisse déduire la caractéristique recherchée. Au passage, il a été également vérifié que les règles qui déduisent la caractéristique voulue n'ont pas été invoquées pour de mauvaises raisons. Pour effectuer cette vérification, TEIRESIAS utilise une structure appelée “modèle de règle” qui spécifie les ensembles de règles capables de conclure sur un attribut donné. TEIRESIAS finit par demander à l'utilisateur de donner une nouvelle règle qui soit capable de déduire l'attribut repéré. L'utilisateur saisit cette règle directement, puis TEIRESIAS l'interprète et la retranscrit dans son vocabulaire.

En générant des exemples que valide ou rejette l'utilisateur, REVINOS évite à celui-ci de devoir entrer “à la main” les caractéristiques de l'exemple courant. REVINOS déduit ces dernières par différenciation comme nous allons le voir par la suite.

b) ASK

Le système ASK ([Gruber, 1989a], [Gruber, 1989b]) est également un assistant interactif d'acquisition des connaissances. ASK contient des règles stratégiques qui lui permettent de choisir les actions pertinentes en fonction de l'état de la Mémoire de Travail, elle-même construite grâce à des connaissances substantives (voir section 2.1.3.2.c). ASK propose à chaque itération un ensemble d'actions à effectuer. L'utilisateur en choisit une et le système l'exécute. L'élicitation des connaissances commence quand l'utilisateur trouve une action à exécuter (appelée exemple positif représentatif) à la place d'une autre figurant dans la sélection du système (appelée exemple négatif représentatif). L'utilisateur doit choisir parmi l'un des trois choix suivants :

- une ou plusieurs actions sont préférées aux autres,
- une ou plusieurs actions ne doivent pas être proposées,
- une action non proposée devrait figurer.

Le système identifie un objectif d'apprentissage à partir des règles stratégiques, de l'exemple positif, de l'exemple négatif et d'heuristiques. L'utilisateur doit alors donner des justifications à son exemple : il sélectionne les objets, propriétés et valeurs dans l'environnement et ASK traduit ces sélections en faits qu'il ajoute dans la liste des justifications. Lorsque les justifications sont satisfaisantes pour remplir l'objectif d'apprentissage, ASK est alors capable d'apprendre la nouvelle règle stratégique.

La même remarque que nous avons faite à propos de TEIRESIAS vaut ici aussi : les caractéristiques de l'exemple courant doivent toujours être données par l'utilisateur, même si l'interface d'ASK rend cette opération plus aisée.

c) ETS

ETS ([Boose, 1985]) est un outil d'élicitation des connaissances basé sur la technique des grilles-répertoires, inspirée des travaux de George Kelly ([Kelly, 1955], [Shaw et Gaines, 1993]). Deux types d'objets apparaissent dans une grille-répertoire : les éléments (exemples) et les attributs, qui vont par paires. L'utilisateur note tous les éléments avec les attributs d'une valeur allant de 1 à 5. L'ensemble de toutes les notes forment une grille. ETS génère alors à partir de la grille un ensemble de règles, ainsi qu'un graphe d'implication. Lorsque l'utilisateur teste un nouveau cas sur le graphe d'implication et qu'il n'est pas satisfait de la conclusion du système, il peut dans un premier temps vérifier la grille. Il peut dans un second temps penser à une exception, donc ajouter un élément, qu'il doit alors noter sur

l'ensemble des attributs. Le graphe d'implication est alors reconstruit. Si cela ne corrige toujours pas le problème, ETS propose à l'utilisateur de raffiner les attributs qui interviennent dans la relation d'implication (par exemple casser une paire d'attributs suspects en deux ou plusieurs paires). Une des fonctionnalités d'ETS est également l'analyse du graphe d'implication pour détecter des éléments conceptuellement proches. ETS invite alors l'utilisateur à les différencier en créant de nouveaux attributs.

Ces différents cas de figure de correction peuvent être mis en correspondance avec ceux de REVINOS de la manière suivante :

- Une correction des notes de la grille et l'ajout d'une exception correspondent à une simple correction de graphe (cas où les faits de la MT sont corrects et complets).

- Le raffinement et la création d'attributs correspond aux cas d'une information manquante ou d'une information erronée.

Les successeurs d'ETS sont les systèmes AQUINAS [Boose, Bradshaw, Koszarek et Shema, 1993] et WEBGRID ([Gaines et Shaw, 1996], [Shaw et Gaines, 1996]).

3.2.2.4 Le problème des choix différés

Nous avons déjà mentionné les stratégies que nous avons adoptées à divers reprises lorsque nous avons été confronté à un choix "à l'aveugle".

Deux effets négatifs peuvent avoir lieu si le système ne prend pas la bonne décision :

1) Si le système prend une mauvaise décision au cours de la session de révision, le modèle (la BC) qui résulte de ce choix sera erronée et il faudra revenir en arrière. Un des principes sur lequel le projet KEW déjà évoqué dans la section 2.6.2 a été construit est la "stratégie d'obligation reportée" (*delayed commitment strategy*) qui consiste à toujours éviter de prendre une décision hasardeuse :

"L'ingénieur de la connaissance ne doit choisir un modèle particulier que s'il y a suffisamment de signes que ce soit là le 'bon modèle'. Ceci reflète l'idée que revenir en arrière dans la construction de modèles abstraits est difficile et doit être évité"

[van Heijst, Terpstra, Shadbolt et Wielenga, 1992, p. 117]

2) D'autre part si le système ne prend pas de décision, cela aura pour effet de complexifier l'apprentissage :

"L'inconvénient majeur de la validation ultérieure est qu'elle conduit le système à manipuler des connaissances inutiles jusqu'à ce qu'elles puissent être supprimées."

[Nédellec, 1994, p. 224]

Le système KRUST ([Craw et Sleeman, 1990], [Craw et Sleeman, 1995]) est une illustration de cette stratégie de report des choix de révision. KRUST construit puis maintient simultanément l'ensemble des BCs qui correspondent aux différentes opérations de révision possibles. Les systèmes SEEK2 ([Ginsberg, 1988a]) et MOBAL ([Morik, Wrobel, Kietz et Emde, 1993]) procèdent de manière similaire. Dans un second temps, la "meilleure" BC est choisie sur des critères de "minimalité" de révision ([Wogulis et Pazzani, 1993], [Wrobel, 1996]). Nous reviendrons sur ce point dans la section 3.6.

Comme tous les systèmes de révision interactive, REVINOS a la possibilité de consulter l'utilisateur pour effectuer les choix qui se posent en temps utile et réduire ainsi la complexité de la révision. La section suivante décrit comment REVINOS parvient à consulter l'utilisateur pour choisir un opérateur de révision.

3.3 Etape 2 : la correction du graphe

Dans cette section nous nous penchons sur l'étape 2 de l'algorithme de la session de révision (figure 3.1). Cette étape est déclenchée lorsque l'étape 1 aboutit à l'identification de la cause d'erreur ou du type de réparation a) ("situation inédite") de la section 3.2[6].

Cette étape est appelée "correction du graphe" parce qu'elle consiste à ajouter ou modifier un lien entre deux nodules qui seront notés par la suite <nodDépart> et <nodCible>. Plus précisément, la correction du graphe est en fin de compte une correction des "règles de choix" du nodule <nodDépart>. Lorsque l'on arrive à cette étape 2, les deux nodules <nodDépart> et <nodCible> existent déjà : ils possèdent un identifiant et une action leur est attachée (<nodCible> peut éventuellement être un nodule tout nouvellement créé dans l'étape 1).

Dans un tout premier temps (section 3.3.1), il s'agit donc de bien préciser cet *objectif de correction*, qui est un triplet (<nodDépart>, <nodCible>, <exCourant>) et dont la signification est la suivante : "pour l'exemple courant <exCourant>, le but de la correction est de faire en sorte qu'après <nodDépart>, ce soit le nodule <nodCible> qui soit activé".

---

[6] Cette cause d'erreur correspond dans le graphe "REVINOS" au nodule "corrigeGraphe".

Dans un second temps (section 3.3.2), il s'agit de sélectionner et appliquer un opérateur de révision qui permette de remplir cet objectif de correction. Lors de cette opération, REVINOS peut proposer d'élargir l'objectif de correction à d'autres exemples similaires.

3.3.1 Identification d'un objectif de correction

L'essentiel du travail d'identification de l'objectif de correction a été fait lors de l'étape 1. Ainsi, ou bien <nodCible> a été déjà repéré, ou bien il a été spécifiquement créé (nodule “nouveauNodule?”).

La question à traiter se réduit donc à l'identification de <nodDépart>. Celui-ci n'est pas toujours le nodule qui était activé au moment précis de l'interruption de la simulation. En effet, rappelons que deux cas de figure se posent : le cas d'une “simulation interrompue” et le cas d'une “simulation terminée” (voir section 3.2.1.1).

Le cas d'une simulation terminée correspond au cas où l'action du nodule courant ayant été complètement exécutée, aucun nodule successeur n'a été trouvé. Dans ces circonstances, <nodCible> doit être attaché à la suite du nodule courant : <nodDépart> est bien le nodule courant.

Dans le cas d'une simulation interrompue, l'action du nodule courant ne convient pas à l'utilisateur. A la fin de l'étape 1, l'utilisateur a précisé que c'était plutôt l'action de <nodCible> qu'il convenait d'exécuter pour l'exemple courant. C'est donc que *depuis le nodule père* du nodule courant, <nodCible> doit être activé à la place du nodule courant. Par conséquent <nodDépart> doit être le dernier nodule activé avant le nodule courant.

Le sous-graphe intitulé “corrigerGraphe” est appelé depuis le nodule “corrigeGraphe” (figure 3.3). Ce sous-graphe remplit la tâche d'identifier l'objectif de correction conformément aux règles que nous venons d'indiquer. Le graphe “corrigerGraphe” ne comporte qu'un seul nodule terminal : le nodule “corrigeRègles”, qui fait lui-même appel au sous-graphe “corrigerRègles” et dont la tâche consiste à appliquer un opérateur de révision. On peut ainsi noter qu'une très grande part du savoir-faire de révision a été explicité sous la forme de trois graphes : “REVINOS”, “corrigerGraphe” et “corrigerRègles”.

3.3.2 Application d'un opérateur de révision

Le but de cette phase est de choisir et d'appliquer un opérateur qui réponde aux objectifs de la correction définis au cours de l'étape précédente. Cette formulation amène deux remarques :

1) L'objectif de correction établi dans la phase précédente est une "exigence minimale" : il est tout à fait possible que REVINOS élargisse la correction, qui a été provoquée par l'exemple courant, à d'autres exemples "proches".

2) Cette phase consiste à choisir un opérateur *parmi* un ensemble d'opérateurs de révision connus. Notre travail nous a mené à identifier trois opérateurs "primitifs" que nous présenterons. Ces opérateurs ont des niveaux de généralité variables, au sens où leur application peut parfois *étendre la correction* (c'est-à-dire faire en sorte que le nodule cible soit activé) à d'autres exemples que l'exemple courant.

Ces deux remarques nous amènent, en tant que concepteur de l'outil de révision coopérative, à prendre une décision dont la problématique est similaire à celle qui se posait dans la section 3.2.1.1 : consulte-t-on ou non l'utilisateur pour effectuer le choix entre plusieurs opérateurs lorsque ce choix se pose ? Et si oui, de quelle manière ?

Contrairement au problème qui se posait dans la section 3.2.1.1, il se trouve que pour notre problème actuel, découvrir plus tard (à l'étape 3) que le choix de l'opérateur n'était pas adéquat oblige vraiment à revenir en arrière exactement à cette étape. Autrement dit, l'erreur ne peut pas véritablement être "rattrapée" sans annuler tout un pan de la session de révision. Pour cette raison, à laquelle s'ajoute notre préférence naturelle pour une consultation de l'utilisateur, nous avons cherché à consulter ce dernier avant de choisir un opérateur, lorsque ce choix se posait.

La première solution adoptée a été la suivante. L'utilisateur était questionné par le système à un niveau "implémentation", c'est-à-dire très proche du problème technique à résoudre par le système (le choix entre deux opérateurs). Sans détailler immédiatement les deux opérateurs concernés par le choix en question, nous pouvons résumer le problème en précisant que la différence essentielle entre les deux opérateurs est la prise en compte ou non, dans la règle modifiée d'un fait, qu'on notera <faitDiscriminant>, qui est une particularité de l'exemple courant. Concrètement, la question qui se pose est la suivante : y-a-t-il ou non un fait discriminant constitutif de la situation courante[7] ? Nous avons donc tenté de poser cette question à l'utilisateur en la formulant de la manière suivante : "Y-a-t-il, dans la

[7] Nous employons l'expression "situation courante" pour qualifier l'ensemble des exemples pour lesquels il faut activer <nodCible>...

liste ci-contre, une information discriminante qui explique que dans la situation actuelle il faille exécuter l'action b plutôt que l'action a ?". Une première manière de régler ce problème a donc été de consulter l'utilisateur ainsi.

La seconde solution, qui finalement a remplacé la première, est beaucoup plus simple. Elle nécessite plus de calcul de la part du système, mais elle est pour l'utilisateur incomparablement plus simple, parce qu'elle pose une question beaucoup plus concrète. Cette seconde solution est la suivante : lorsque plusieurs opérateurs peuvent permettre de corriger la BC afin de remplir l'objectif de correction, REVINOS soumet des exemples à l'utilisateur avant d'effectuer son choix. Cette solution est conforme à la stratégie utilisée par APT et à notre démarche de "révision située" :

"Utilisez autant que possible des cas concrets qui sont plus faciles à évaluer pour l'expert"

[Nédellec et Causse, 1992, p. 173]

Les exemples soumis au cours de l'étape 2 sont calculés de la même manière que les exemples soumis pour validation au cours de l'étape 3. C'est le calcul de la couverture du graphe, avant et après sa correction, qui permet de générer ces exemples. Nous allons détailler immédiatement cette notion de couverture du graphe, essentielle ensuite dans notre processus de validation des révisions effectuées.

3.3.2.1 Couverture d'un graphe

La couverture d'un graphe est l'ensemble de tous les *parcours* possibles du graphe, exprimés à un certain degré d'abstraction. Chaque parcours est un exemple qu'on qualifiera “exemple abstrait”, exprimé sous la forme d'une liste de faits *abstraits* (voir figure 3.14) renvoyés par les actions simulées au cours de ce parcours.

fait déclaré (ce que peut renvoyer l'action a)	attribut1 CHAINE
fait abstrait (calculé pour la couverture du graphe g)	attribut1 != 'Lyon' ET attribut1 != 'Valence'
fait instancié (stocké dans la MT au cours d'une exécution)	attribut1 = 'Marseille'

Figure 3.14 : Différents degrés d'abstraction des faits utilisés dans REVINOS

Un parcours peut être représenté par une suite ordonnée de faits abstraits. Pour des raisons de lisibilité, avant chaque fait, est indiqué le nom de la procédure ou de l'écran concerné (voir figure 3.15). L'ordre des faits est celui des actions exécutées. L'algorithme de calcul de couverture simule donc l'exécution du graphe et enrichit la liste des exemples abstraits à partir des descriptions des faits des actions rencontrées. Pour les faits qui sont des attributs valués, les intervalles de valeurs possibles sont mis à jour au fur et à mesure du parcours du graphe, au vu des règles activées pour choisir les nodule successeurs.

Le calcul de couverture des graphes est rendu possible par le fait que :

- d'une part toute action doit avoir été déclarée et dans cette déclaration sont décrits les faits que l'action renvoie,
- d'autre part, la BC est découpée en graphes acycliques modulaires et l'on peut ainsi limiter le calcul de la couverture au niveau d'un seul graphe.

benefFacCir(agent),facCirCP().
benefFacCir(ascendant),facCirFamille().
benefFacCir(conjoint),facCirCarteAyantDroit().
benefFacCir(enfant),origine(enfant.origine.agent),celibataire(),pasDeFacCir().
benefFacCir(enfant),origine(enfant.origine.agent),celibataire(enfant.celibataire),
age>21ans(enfant.age<21ans),facCirCarteAyantDroit().
benefFacCir(enfant),origine(enfant.origine.agent),celibataire(enfant.celibataire),
age>21ans(enfant.age>21ans),casParticulierEnfantDe21ans(enfant.aCharge),facCirFamille().
benefFacCir(enfant),origine(enfant.origine.agent),celibataire(enfant.celibataire),
age>21ans(enfant.age>21ans),casParticulierEnfantDe21ans(enfant.auServiceNational),
facCirEnfantSNA().
benefFacCir(enfant),origine(enfant.origine.agent),celibataire(enfant.celibataire),
age>21ans(enfant.age>21ans),casParticulierEnfantDe21ans(enfant.etudiant),conditionsFacCir().
benefFacCir(enfant),origine(enfant.origine.agent),celibataire(enfant.celibataire),
age>21ans(enfant.age>21ans),casParticulierEnfantDe21ans(enfant.malade),conditionsFacCir().
benefFacCir(enfant),origine(enfant.origine.concubin),celibataire(),pasDeFacCir().
benefFacCir(enfant),origine(enfant.origine.concubin),celibataire(enfant.celibataire),
age>21ans(enfant.age<21ans),facCirEnfant().
benefFacCir(enfant),origine(enfant.origine.concubin),celibataire(enfant.celibataire),
age>21ans(enfant.age>21ans),casParticulierEnfantDe21ans(enfant.aCharge),facCirFamille().
benefFacCir(enfant),origine(enfant.origine.concubin),celibataire(enfant.celibataire),
age>21ans(enfant.age>21ans),casParticulierEnfantDe21ans(enfant.auServiceNational),
facCirEnfantSNA().
benefFacCir(enfant),origine(enfant.origine.concubin),celibataire(enfant.celibataire),
age>21ans(enfant.age>21ans),casParticulierEnfantDe21ans(enfant.etudiant),facCirEnfant().
benefFacCir(enfant),origine(enfant.origine.concubin),celibataire(enfant.celibataire),
age>21ans(enfant.age>21ans),casParticulierEnfantDe21ans(enfant.malade),facCirEnfant().
benefFacCir(enfant),origine(enfant.origine.recueilli),confie(enfant.confié),celibataire(),pasDeFacCir().
benefFacCir(enfant),origine(enfant.origine.recueilli),confie(enfant.confié),celibataire(enfant.celibataire),
age>21ans(enfant.age<21ans),conditionsFacCir().
benefFacCir(enfant),origine(enfant.origine.recueilli),confie(enfant.confié),celibataire(enfant.celibataire),
age>21ans(enfant.age>21ans),casParticulierEnfantDe21ans(enfant.aCharge),facCirFamille().
benefFacCir(enfant),origine(enfant.origine.recueilli),confie(enfant.confié),celibataire(enfant.celibataire),
age>21ans(enfant.age>21ans),casParticulierEnfantDe21ans(enfant.auServiceNational),
facCirEnfantSNA().
benefFacCir(enfant),origine(enfant.origine.recueilli),confie(enfant.confié),celibataire(enfant.celibataire),
age>21ans(enfant.age>21ans),casParticulierEnfantDe21ans(enfant.etudiant),conditionsFacCir().
benefFacCir(enfant),origine(enfant.origine.recueilli),confie(enfant.confié),celibataire(enfant.celibataire),
age>21ans(enfant.age>21ans),casParticulierEnfantDe21ans(enfant.malade),conditionsFacCir().
benefFacCir(enfant),origine(enfant.origine.recueilli),confie(enfant.nonConfié),ageRecueillement(),
pasDeFacCir().
benefFacCir(enfant),origine(enfant.origine.recueilli),confie(enfant.nonConfié),
ageRecueillement(enfant.recueilliAvant21ans),celibataire(),pasDeFacCir().
benefFacCir(enfant),origine(enfant.origine.recueilli),confie(enfant.nonConfié),
ageRecueillement(enfant.recueilliAvant21ans),celibataire(enfant.celibataire),
age>21ans(enfant.age<21ans),facCirEnfant().
benefFacCir(enfant),origine(enfant.origine.recueilli),confie(enfant.nonConfié),
ageRecueillement(enfant.recueilliAvant21ans),celibataire(enfant.celibataire),
age>21ans(enfant.age>21ans),pasDeFacCir().

Figure 3.15 : La couverture[8] du graphe “facilités de circulation”

[8] Cette figure affiche la couverture du graphe exactement telle que REVINOS la génère. Les faits sont séparés par des virgules, les parcours sont séparés par des points.

3.3.2.2 Exemples partiels

La couverture d'un graphe G, qu'on peut noter Couv(G), contient donc l'ensemble des parcours possibles de ce graphe à partir du nodule initial. On peut également définir la notion de "**couverture d'un nodule n**" (Couv(n)), où n appartient au graphe G, comme le sous-ensemble de Couv(G) des parcours du graphe G passant par n.

Couv(n) est intéressant parce que c'est une représentation de tous les cas résolus (une situation concrète plus une action finale) "concernés" par le nodule n. Inversement, une modification du contenu du nodule n peut modifier les solutions proposées par G à toutes les situations concrètes de Couv(n).

La notion de "**couverture partielle d'un nodule n**", que nous noterons CouvPart(n) va s'avérer très utile pour remplir notre objectif de correction des règles de choix d'un nodule. Au lieu de développer jusqu'à son terme chaque parcours après activation de n, l'algorithme de calcul de CouvPart(n) se contente d'indiquer le nom du nodule activé juste après n.

CouvPart(n) permet ainsi de calculer des **exemples partiels**, qui peuvent être affichés à l'utilisateur de manière à aider au choix de l'opérateur de révision approprié. La figure 3.16 montre un exemple partiel calculé à partir de CouvPart(nodDépart). L'action affichée est celle du nodule successeur du nodule <nodDépart>. La réponse du concepteur à ce type de question permet à REVINOS de savoir s'il doit étendre la correction effectuée pour l'exemple initial à des exemples "proches".

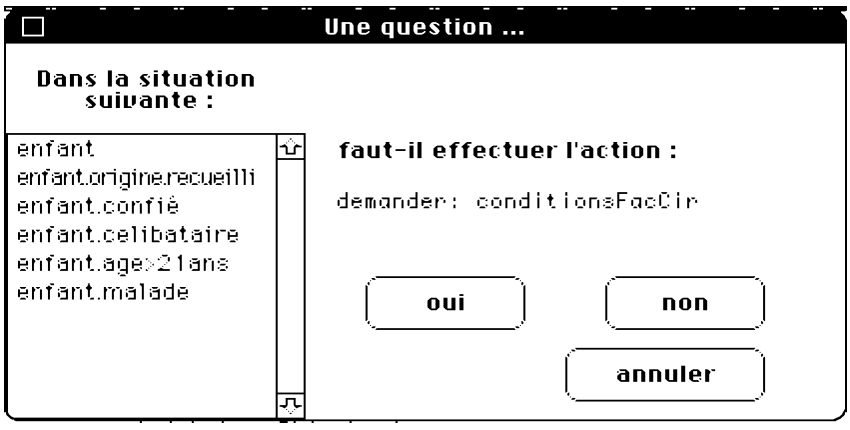


Figure 3.16 : Soumission d'un exemple partiel

3.3.2.3 Choix d'un opérateur

Trois opérateurs de correction des règles de choix ont été définis. Le premier opérateur *créeRègle* consiste à simplement ajouter à l'ensemble des règles de choix de <nodDépart>, une nouvelle règle qui mène à <nodCible>. Cet opérateur est appliqué lorsqu'aucune règle de choix dans <nodDépart> n'est activable pour l'exemple courant.

Les deux autres opérateurs corrigent la règle activée pour l'exemple courant. Cette règle menait au nodule qu'on appellera par la suite <ancienNod> pour l'exemple courant.

L'opérateur *modifieRègle* procède à une simple substitution dans la règle activée, de <ancienNod> par <nodCible>. Il propose donc d'étendre la correction courante (activer <nodCible> plutôt que <ancienNod>) à l'ensemble des exemples qui utilisent la règle activée. C'est donc un opérateur de généralisation.

L'opérateur *remplaceRègle* est plus spécifique et remplace la règle activée par deux nouvelle règles, l'un orientant sur <nodCible>, l'autre sur <ancienNod>.

REVINOS utilise les exemples partiels soumis à l'utilisateur (figure 3.16) :

1) pour choisir entre les deux opérateurs *remplaceRègle* et *modifieRègle,*

2) lors de l'application de l'opérateur *remplaceRègle,* pour calculer, par différenciation, les faits discriminants qui orientent soit sur <ancienNod>, soit sur <nodCible>.

La figure 3.17 détaille l'algorithme utilisé pour choisir l'opérateur approprié.

```
si aucune règle n'est activable pour l'exemple courant,
        appliquer l'opérateur créeRègle
sinon
        repérer la règle activée par l'exemple courant  (<ancienNod> est le nodule
          activé par cette règle)
        soumettre à l'utilisateur tous les exemples "partiels" qui utilisent  cette règle
          pour passer à  <ancienNod>
        si tous ces exemples sont refusés
                appliquer l'opérateur modifieRègle
        sinon (un des exemples a été accepté)
                appliquer l'opérateur remplaceRègle
        finsi
finsi
```

Figure 3.17 : Algorithme pour l'application d'un opérateur

L'opérateur *créeRègle* est décrit dans la figure 3.18. Pour appliquer cet opérateur, il est nécessaire de connaître le fait renvoyé par l'action de <nodDépart> pour l'exemple courant. Ce fait, noté <fait1>, constitue la prémisse de la nouvelle

règle, sauf dans le cas où il vaut nil : dans ce cas l'opérateur a besoin de savoir quels autres faits (dits *faits alternatifs*) peut renvoyer l'action de <nodDépart>.

```
repérer <fait1>, le fait renvoyé par l'action de <nodDépart>
si <fait1> n'est pas vide
        ajouter la règle 'si: <fait1> alorsAller: <nodCible>'
sinon
        repérer le fait alternatif <alterFait> de <fait1>
        s'il n'y a qu'un seul fait alternatif <fait2>
                ajouter la règle 'si: NON(<fait2>) alorsAller: <nodCible>
        sinon
                ajouter la règle 'aller: <nodCible>' avec la priorité la plus faible
        finsi
 finsi
```

Figure 3.18 : Algorithme de l'opérateur *créeRègle*

L'opérateur *modifieRègle* est décrit dans la figure 3.19.

```
repérer la règle activée, sa <prémisse> et sa priorité <p>
retirer la règle activée des règles du nodule <nodDépart>
ajouter la règle 'si: <prémisse> alorsAller: <nodCible > priorite: <p>'
```

Figure 3.19 : Algorithme de l'opérateur *modifieRègle*

L'opérateur *remplaceRègle* , décrit dans la figure 3.20, est appliqué lorsque l'on connaît un exemple partiel, qu'on notera <exSoumis>, pour lequel l'utilisateur a confirmé que l'exécution de l'action de <ancienNod> est bien pertinente (l'utilisateur a répondu non à une question du type de celle apparaissant dans la figure 3.16). Autrement dit, la correction pour <exCourant> ne doit pas être étendue à <exSoumis>. Les états la MT pour <exCourant> et <exSoumis> sont comparés, afin d'identifier ce que l'on appelle des faits discriminants, qui seront utilisés dans les prémisses des deux règles qui remplaceront la règle activée. Trois cas de figure peuvent se présenter :

- des faits propres à chaque exemple sont présents de chaque côté. Deux ensembles de faits discriminants sont donc trouvés. Ils permettent de créer très facilement les deux règles adéquates. Ainsi, si par exemple <exSoumis> = (a, b, c) et <exCourant> = (c, d, e), on a les deux ensembles de faits discriminants (a, b) et (d, e), qui vont permettre de construire les règles “si: <prémisse> ET a ET b alorsAller: <ancienNod> priorite: <p>” et “si: <prémisse> ET d ET e alorsAller: <nodCible> priorite: <p>”[9].

[9] Où <prémisse> est la prémisse de la règle activée pour l'exemple courant.

- des faits discriminants sont trouvés uniquement dans <exCourant>. REVINOS utilise un de ces faits discriminants, <faitD>, pour créer la règle qui activera <nodCible>. La seconde règle sera créée à partir du "fait alternatif" <alterFait> de <faitD>, qui est calculé d'après la déclaration de l'action <a> ayant renvoyé <faitD>. Dans l'algorithme de la figure 3.20 <alterFait> est vide si <a> peut renvoyer alternativement à <faitD> plusieurs autres faits. Il vaut NON(faitD) si la seule alternative à <faitD> est nil.

- des faits discriminants sont trouvés uniquement dans <exSoumis>. La démarche est alors la même que pour le cas précédent sauf qu'il y a l'inversion suivante : la règle qui comprendra <faitD> activera alors <ancienNod>.

```
repérer la règle activée, sa <prémisse> et sa priorité <p>
retirer la règle activée des règles de <nodDépart>
comparer les MT de <exSoumis> et de <exCourant>
si l'on peut identifier deux ensembles non vides, <faitsDiscrimSoumis> et
  <faitsDiscrimCourant> de faits propres à (resp.) <exSoumis> et  <exCourant>,
        ajouter la règle 'si: <prémisse> ET <faitsDiscrimSoumis> alorsAller:
          <ancienNod>  priorite: <p>'
        ajouter la règle 'si: <prémisse> ET <faitsDiscrimCourant> alorsAller:
          <nodCible>  priorite: <p>'
sinon
        soit le fait discriminant <faitD>
        si <faitD> était dans <exSoumis>,
                nodVisé := ancienNod
                alterNod := nodCible
        finsi
        si <faitD> était dans <exCourant>
                nodVisé := nodCible
                alterNod := ancienNod
        finsi
        ajouter la règle 'si: <prémisse> ET <faitD> alorsAller: <nodVisé >
          priorite: <p>'
        repérer le fait alternatif <alterFait> de <faitD>
        si <alterFait> est vide (plusieurs autres faits peuvent être renvoyés
alternativement à <faitD>),
                ajouter la règle 'si: <prémisse> alorsAller: <alterNod>'
                  avec une priorité plus faible que <p>
        sinon
                ajouter la règle 'si: <prémisse> ET <alterFait>
                  alorsAller: <alterNod> priorite: <p>'
        finsi
finsi
```

Figure 3.20 : Algorithme de l'opérateur *remplaceRègle*

3.3.3 Quelques exemples de correction d'un graphe

3.3.3.1 Exemple d'application de l'opérateur *créeRègle*

En poursuivant la session de révision que nous avons décrite dans la section 3.2.1.4, l'étape 1 appliquée à l'exemple que nous avons appelé “exempleCOD” (figure 3.13), va amener le concepteur à créer un nodule “accordPourPronomRéfléchiCOD” qui exécute l'action “demander accordAvecLeSujet”. L'objectif de correction pour l'étape 2 est alors le suivant : <nodDépart> est <verbePronominal>, <nodCible> est <accordPourPronom-RéfléchiCOD> et <exCourant> est <exempleCOD>.

Comme le nodule “verbePronominal” ne contient pas de règles de choix (c'était jusqu'à présent un nodule terminal), l'opérateur *créeRègle* va être appliqué et donnera la règle suivante : “si: pronomReflechiCOD alorsAller: pronomRéfléchiCOD ”.

De manière exactement similaire, l'intégration de l'exemple initial “ils se sont succédé” lors de la troisième révision aboutira à l'ajout d'une deuxième règle pour le nodule “verbePronominal” : “si: pronomReflechiCOI alorsAller: pronomRéfléchiCOI”. Rappelons que le graphe ainsi révisé est loin d'être complet. L'annexe 4 affiche le compte-rendu de la session de révision complète.

3.3.3.2 Exemples d'application des opérateurs *modifieRègle et remplaceRègle*

Le graphe de la figure 3.21 a pour but de donner les facilités de circulation auxquelles peuvent prétendre les différentes catégories d'ayant-droit. Les membres de la famille d'un agent S.N.C.F. peuvent en effet bénéficier sous certaines conditions de coupons de voyages gratuits ou à prix réduit.

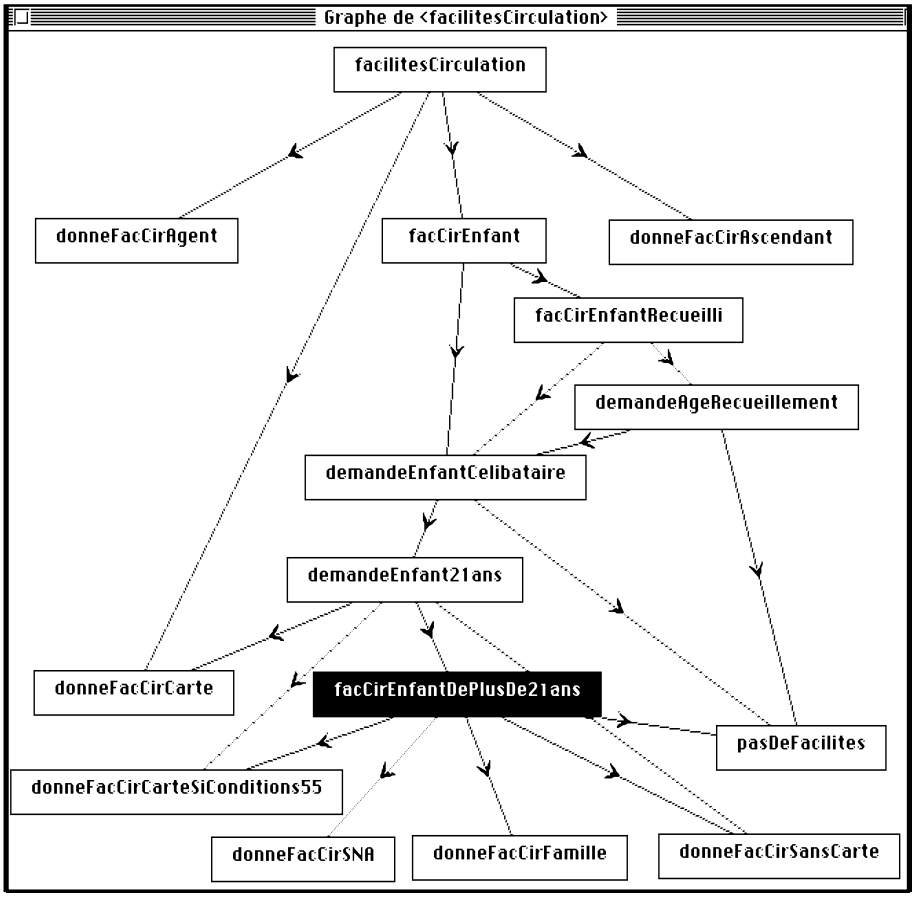


Figure 3.21 : Graphe “facilitésDeCirculation”

Le nodule “facCirEnfantDePlusDe21ans” correspond à la situation où le bénéficiaire est un enfant célibataire âgé de plus de 21 ans. L'unique action de ce nodule est un appel à l'écran “casParticulierEnfantDePlusDe21ans” qui demande au minitéliste de préciser à quelle catégorie donnant lieu à des facilités de circulation appartient l'enfant:

- l'enfant est étudiant âgé de moins de 26 ans (fait “enfant.etudiant”),
- l'enfant est atteint d'une maladie de longue durée ou d'une invalidité (fait “enfant.malade”),
- l'enfant effectue son service national (fait “enfant.auServiceNational”),
- l'enfant est à la charge de ses parents (fait “enfant.aCharge”).

Les règles de choix du nodule “facCirEnfantDePlusDe21ans” apparaissent dans la figure 3.22.

```
si: enfant.origine.concubin ET (enfant.etudiant OU enfant.malade) alorsAller:
        donneFacCirSansCarte priorite: 1;
si: enfant.etudiant alorsAller: donneFacCirCarteSiConditions55 priorite: 2;
si: enfant.malade alorsAller: donneFacCirCarteSiConditions55' priorite: 2;
si: enfant.aCharge alorsAller: donneFacCirFamille;
si: enfant.auServiceNational alorsAller: donneFacCirSNA
```

Figure 3.22 : Les règles de choix du
nodule “facCirEnfantDePlusDe21ans” avant révision

L'exemple courant pour lequel le graphe “facilitésDeCirculation” doit être corrigé est celui d'un enfant d'agent, infirme, célibataire, âgé de plus de 21 ans, vivant à la charge de ses parents. Pour ce cas précis, qu'on appellera donc <exCourant> par la suite, la première étape de révision conduit le concepteur à créer un nouveau nodule “donneFacCirCarteSiConditions75” dans lequel il est annoncé au minitéliste que sous réserve d'avoir des ressources inférieures à 75 % du SMIC, le bénéficiaire a droit à un certain nombre de facilités de circulation précisées par ailleurs. L'objectif de correction est donc le suivant : <nodDépart> est <facCirEnfantDePlusDe21ans>, <nodCible> est <donneFacCirCarteSi-Conditions75> et <exCourant> a été décrit ci-dessus (la figure 3.24 fait apparaître la description de la MT pour <exCourant>).

Dans ce contexte, <ancienNod> est <donneFacCirCarteSiConditions55>. REVINOS détecte un exemple partiel qu'il soumet à l'utilisateur : c'est l'exemple qui était affiché dans la figure 3.16. Le cas d'un enfant recueilli, confié sur décision de justice, infirme, célibataire, âgé de plus de 21 ans, vivant à la charge de ses parents, doit-il être traité exactement de la même manière que <exCourant> ? La réponse à cette question n'apparaît d'ailleurs pas explicitement dans le document ayant servi de support à la construction du graphe, ce qui montre que REVINOS est utile à l'élicitation des connaissances...

Si le concepteur rejette l'exemple soumis en répondant par la négative à la question de la figure 3.16, <exSoumis> et <exCourant> doivent tous les deux aboutir au nodule “donneFacCirCarteSiConditions75”. L'opérateur *modifieRègle* remplace la règle “si: enfant.etudiant alorsAller: donneFacCirCarteSiConditions55 priorite: 2;” par la règle “si: enfant.etudiant alorsAller: donneFacCirCarteSiConditions75 priorite: 2;”.

Si le concepteur décide que la correction ne doit être faite que pour <exCourant> et que le traitement de <exSoumis> ne doit pas être changé, REVINOS applique l'opérateur *remplaceRègle*. Le fait discriminant est <enfant.origine.recueilli> et les nouvelles règles de choix du nodule apparaissent en

gras dans la figure 3.23. L'annexe 5 affiche le compte-rendu de la session de révision complète.

```
si: enfant.origine.concubin ET (enfant.etudiant OU enfant.malade) alorsAller:
        donneFacCirSansCarte priorite: 1;
si: enfant.etudiant alorsAller: donneFacCirCarteSiConditions55 priorite: 2;
si: enfant.malade ET enfant.origine.agent alorsAller:
        donneFacCirCarteSiConditions75 priorite: 2;
si: enfant.malade alorsAller: donneFacCirCarteSiConditions55
        priorite: 3;
si: enfant.aCharge alorsAller: donneFacCirFamille;
si: enfant.auServiceNational alorsAller: donneFacCirSNA
```

Figure 3.23 : Les règles de choix du nodule “facCirEnfantDePlusDe21ans” après révision

3.4 Etape 3 : validation de la révision

Lorsque la correction a été effectuée par l'application d'un opérateur, le système en calcule toutes les répercussions et les présente à l'utilisateur comme autant d'exemples à valider. Les répercussions d'une modification sont calculées en faisant la différence entre la couverture du graphe avant la correction et la couverture du graphe après la correction. Une répercussion est donc un “exemple abstrait” ou “parcours” conformément au vocabulaire introduit dans la section 3.3.2.2.

Deux ensembles sont constitués :

E+ = nouveaux exemples qui n'étaient auparavant pas couverts.

E- = exemples qui ne sont désormais plus couverts.

L'ensemble E+ est ajouté à la liste L des exemples à valider, qui contenait au départ de la révision courante, au moins l'exemple courant. L'ensemble E- est utilisé pour mettre à jour L et en retirer les exemples qui seraient maintenant caduques. Une liste L' des exemples déjà validés est également maintenue. Elle sert notamment à éviter de soumettre à l'utilisateur, lors de l'étape 2, des exemples partiels déjà validés.

La figure 3.24 montre le type de question posée à l'utilisateur à cette étape, identique à la soumission d'exemples partiels (figure 3.16). A cette étape l'utilisateur s'attend à confirmer des situations pour lesquelles il s'est déjà prononcé. Par exemple, il reverra des situations que REVINOS lui avait déjà soumises sous forme partielle, qui apparaîtront maintenant sous la forme de plusieurs cas résolus. L'utilisateur valide en fait les conséquences de ses choix antérieurs.

Comme nous l'avons montré dans la section 3.2.1.4, cette étape peut être le point de départ d'une nouvelle révision.

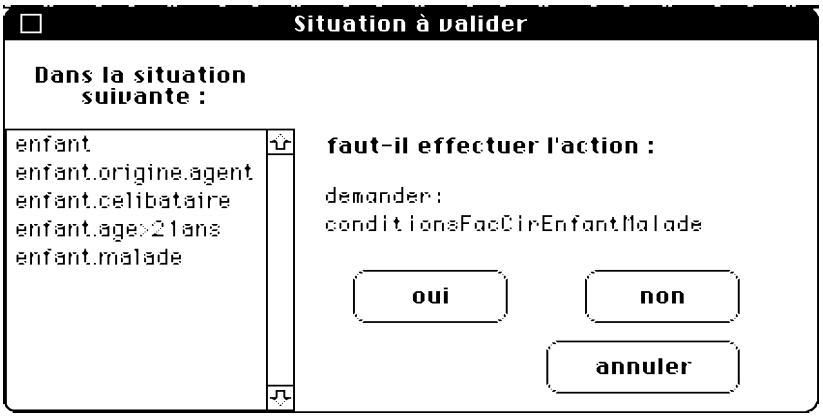


Figure 3.24 : Validation d'une répercussion

3.5 Etape 4 : recherche d'une généralisation supplémentaire

Une fois que toutes les répercussions ont été validées, le système peut proposer, s'il en trouve, des généralisations de la correction apportée.

La généralisation qui a été implémentée dans la version courante de REVINOS s'appuie sur les champs “règles de choix” des nodules qui ont été révisés par un opérateur *remplaceRègle* ou *modifieRègle*. Cette généralisation est possible uniquement si d'autres règles de choix des nodules corrigés mènent aussi à <ancienNod>. Le principe est de soumettre à l'utilisateur les exemples qui utilisent ces autres règles pour chercher si retoucher une de ces règles par un des deux opérateurs ne serait pas pertinent. Si c'est le cas, on étend alors la révision à des situations proches, qui n'avaient pas été prises en compte initialement.

Dans l'exemple des facilités de circulation, il existe une autre règle de choix du nodule ”facCirEnfantDePlusDe21ans” qui mène à 'donneFacCirCarteSiConditions55' (voir figure 3.23). Il s'agit de la règle : “si: enfant.etudiant alorsAller: donneFacCirCarteSiConditions55 priorite: 2;”.

Il paraît pertinent de demander à l'utilisateur si ”donneFacCirCarteSiConditions75” ne doit pas être aussi activé pour les enfants étudiants. REVINOS demande donc au concepteur à la fin de la session de révision s'il souhaite chercher à généraliser les corrections effectuées. Si le concepteur répond par l'affirmative, REVINOS lui soumettra l'exemple d'un étudiant fils

d'agent. Et le concepteur pourra alors ou non décider de déclencher une révision pour cet exemple.

Le principe de chercher à généraliser une correction est utile au concepteur, pour deux raisons :

- D'abord parce que cela peut aider le concepteur à corriger des erreurs similaires à celle corrigées précédemment. L'extension de la correction recherchée dans un premier temps parmi tous les parcours qui aboutissent à <ancienNod> en passant par <nodDépart>, peut être ensuite tentée pour l'ensemble des parcours du graphe qui aboutissent à <ancienNod>. REVINOS pourrait donc aussi passer en revue avec le concepteur l'ensemble de ces parcours.

- Plus généralement, dans sa démarche d'acquisition incrémentale des connaissances, le concepteur est toujours intéressé par la découverte de nouvelles situations concrètes à modéliser, REVINOS peut donc aider à détecter celles-ci en soumettant un exemple “incomplet”. Dans cette optique, essayer de généraliser le nodule <nodCible>, c'est-à-dire trouver de nouveaux parcours pour <nodCible> et par extension pour un nodule quelconque, est une opération qui peut à tout moment intéresser le concepteur, même si cela peut avoir un rapport assez lointain avec la dernière correction effectuée. Cette direction pourra être explorée lors d'un prochain enrichissement de REVINOS.

D'une manière générale, le concepteur a toute liberté de choisir entre poursuivre cette étape de “détection” d'erreur en cherchant des généralisations éventuelles proposées par REVINOS, et démarrer une session de révision sur un autre exemple repéré plus tôt.

3.6 Travaux similaires

3.6.1 Notion d'arbre de preuve

Les systèmes de révision étant pour la plupart basés sur la logique, ils partagent tous le point commun de toujours appuyer le processus de révision sur l'arbre de preuve. Cet arbre de preuve, souvent présenté sous la forme d'un arbre ET/OU, est ainsi appelé *dérivation* dans le module de révision KRT du système MOBAL ([Morik, Wrobel, Kietz et Emde, 1993]), *proof tree* dans KRUST ([Craw et Sleeman, 1990]), *justification forest* dans WHY ([Saitta, Botta et Neri, 1993]),... L'arbre de preuve sert à connaître exactement quelles informations ont participé à la dérivation de l'inférence incorrecte. C'est une notion qui sert de base aux techniques d'EBG (*Explanation-Based Generalization*, voir section 1.2.2.2), d'évaluation

partielle ([van Harmelen et Bundy, 1988]) et de Programmation Logique Inductive (Inductive Logic Programming ou ILP en anglais).

L'équivalent de l'arbre de preuve dans REVINOS est le parcours courant plus le graphe de nodules lui-même. La grande particularité de REVINOS, c'est donc que la structure centrale sur laquelle l'algorithme de révision se fonde, correspond exactement au formalisme de représentation des connaissances. L'avantage n'est pas simplement d'éviter à chaque fois de reconstruire dynamiquement une *explication*, une *justification* ou une *dérivation*. Le recours à un formalisme unique pour l'exploitation de la BC et sa révision permet surtout de faciliter considérablement la communication entre le système et l'utilisateur. Avec REVINOS, l'utilisateur comprend, réutilise, crée l'équivalent des nœuds de l'arbre de preuve : les nodules de situations. Alors que KRUST, KRT ou SEEK2 sont contraints de construire des ensembles de révisions possibles, et d'ensuite trouver des moyens de sélectionner la meilleure révision, REVINOS réduit considérablement l'espace du problème en définissant de manière coopérative avec l'utilisateur un "objectif de correction" défini dans les termes mêmes de l'arbre de preuve (après le nœud x, aller au nœud y). L'étape 2 de correction dans REVINOS est en conséquence beaucoup plus contrainte, donc plus facilement automatisable dans la mesure où moins de choix importants se posent durant la révision.

3.6.2 La soumission d'exemples

Différentes sources de connaissances peuvent être utilisées pour réviser une BC. On peut citer ainsi :

- le recours à un utilisateur expert du domaine, ou capable de faire l'interface avec un expert du domaine,
- une théorie du domaine prédéfinie dans un formalisme donné,
- un ensemble connu de cas déjà résolus (base d'exemples),
- des méta-connaissances ou "biais d'apprentissage" ([Nédellec, Rouveirol, Adé, Bergadano et Tausend, 1996]),
- un environnement, terrain "d'expérimentation active" ([Carbonnell et Gil, 1990]).

La dernière source de connaissances, l'environnement, ne peut être utilisée que par des systèmes capables de mener à bien des expériences sur l'environnement et d'en interpréter les résultats. PRODIGY ([Carbonnell et Gil, 1990]) rentre dans cette catégorie de systèmes. Un module appelé "Design-PRODIGY" planifie une séquence d'actions pour "perturber" l'environnement. Ensuite le module

"Observation-PRODIGY" est capable "d'instrumentaliser" l'expérimentation ([Carbonell, Knoblock et Minton, 1991]).

La plupart des systèmes de révision, comme SEEK2 ([Ginsberg, 1988a]), EITHER ([Ourston et Mooney, 1990]) ou INBF ([Smith et Rosenbloom, 1990]) ont été conçus pour fonctionner sans utilisateur, à partir uniquement d'une base d'exemples classés. Dans ces circonstances, la "qualité" de l'apprentissage dépend étroitement de la représentativité des exemples préalablement connus.

A l'opposé, le système ASK, présenté dans la section 3.2.2.3, ne dispose pas de base d'exemples mais interagit avec l'utilisateur. L'utilisateur est consulté au cours de cette généralisation, par exemple pour identifier les domaines de validité des variables apparaissant dans la règle apprise. Mais comme le souligne Tom Gruber, comme tout repose sur l'utilisateur, ASK risque de surgénéraliser l'exemple appris.

Avoir sous la main un utilisateur disponible présente un grand intérêt pour un système de révision à cause de la flexibilité et la réactivité que cela apporte au processus de révision, à condition de faire l'hypothèse que l'utilisateur-expert est infaillible. Or il n'est pas toujours évident pour un utilisateur de détecter immédiatement une erreur ou une incomplétude lorsqu'on lui présente une règle générale. Demander à l'utilisateur d'examiner des exemples concrets possède alors l'avantage de limiter le risque d'erreur de la part de l'utilisateur. "L'heuristique des situations spécifiques" (cf. section 1.2.3.2) permet de recenser les différents cas particuliers qui seraient susceptibles d'être oubliés quand l'utilisateur cherche à valider des règles générales.

Ainsi, les systèmes MIS ([Shapiro, 1981]), MARVIN ([Sammut et Banerji, 1986]), CLINT, DISCIPLE, APT ou FILP ([Bergadano et Gunetti, 1993]), par exemple, communiquent avec l'utilisateur en générant des exemples utiles au problème courant d'apprentissage et en demandant à l'utilisateur de les "classer", c'est-à-dire de préciser si l'exemple appartient ou non à la classe que le système veut apprendre. [Angluin, 1988] appelle ce type d'opération, qui retourne un seul bit d'information : requête d'appartenance (*membership queries*). C'est le type de requête le plus simple, par opposition aux autres types de requête que sont les requêtes d'équivalence, de sous-ensemble, de sur-ensemble, etc. Mais c'est aussi le type de requête qui renvoie le moins d'information. Le problème devient alors : comment apprendre correctement sans devoir soumettre de trop nombreux exemples à l'utilisateur ? CLINT par exemple utilise l'opérateur d'abandon de littéraux pour généraliser et il a besoin de plus d'exemples que MARVIN, qui, lui, s'appuie sur l'opérateur d'absorption, et risque en contrepartie davantage la surgénéralisation ([Nédellec, 1994]).

Dans les exemples que présentent MIS, MARVIN et CLINT à l'utilisateur, les faits sont instanciés. Ainsi, un concept essai ("*crucial object*") construit par MARVIN pour tenter de généraliser la définition courante du concept cible (en l'occurrence le concept de *té*), pourrait être formulé de la manière suivante : "Si A est une brique, en position horizontale, si B est une colonne, et si A est posé sur B, l'ensemble A et B forme-t-il un té ?" (d'après [Sammut et Banerji, 1986]).

Les exemples construits par DISCIPLE et APT sont d'un niveau plus abstrait et ressemblent davantage à des règles parce qu'ils contiennent des variables quantifiées universellement. Exemple : "Quels que soient X, Y et Z, si X est une sphère, Y est un bloc, et Z un parallélépipède, est-il possible de construire une arche (en posant Z sur X et Y) ?" (d'après [Nédellec et Causse, 1992]).

Les exemples que construit REVINOS pour les soumettre à l'utilisateur ont un niveau d'abstraction plus élevé que les exemples concrets fabriqués au cours d'une simulation et constitués de faits instanciés. Un exemple "abstrait", constitué de faits "abstraits" (voir figure 3.14) correspond à un ensemble d'exemples concrets ayant pour point commun de produire le même parcours dans le graphe. Cette notion d'exemple abstrait, qui permet de réduire le nombre d'exemples soumis à l'utilisateur, est une particularité notable de REVINOS par rapport aux autres systèmes de révision.

Conformément à la remarque effectuée ci-dessus à propos du système ASK, il semble que l'abstraction des exemples ne doit pas être poussée trop loin. Tous les parcours aboutissant à un même nodule ne sont pas identiques d'un point de vue conceptuel et il n'est pas utile de les regrouper au sein du même exemple abstrait. Ainsi, l'algorithme de calcul de couverture d'un graphe crée autant d'exemples qu'il y a de faits déclarés pour une action donnée, lorsque la déclaration de l'action contient un ensemble fini de valeurs possibles. Par exemple, lors du calcul de la couverture du graphe "facilitésDeCirculation" (figure 3.15), chaque parcours partiel menant au nodule "facCirEnfantDePlusDe21ans" donne quatre nouveaux parcours correspondants aux quatre faits "enfant.etudiant", "enfant.malade", "enfant.aCharge" et "enfant.auServiceNational". En distinguant précisément des faits qui peuvent être alternativement renvoyés, le concepteur distingue du même coup autant de cas de figure différents à prendre en considération. Lorsque le concepteur ne précise pas de domaine de valeurs fini pour un fait, il ne donne aucune indication sur les cas de figure pertinents qui peuvent se présenter. Si le fait <f> renvoyé par l'action du nodule "facCirEnfantDePlusDe21ans" avait été une chaîne de caractère, le nombre d'exemples abstraits finalement générés par un parcours partiel menant au nodule "donneFacCirSiConditions55" aurait dépendu

des tests effectués sur la valeur de <f> dans les règles de choix des nodules suivants.

3.6.3 Comparaison des opérateurs

Deux types de révision peuvent être distingués :

- Le premier type de révision s'appuie sur une structure construite spécifiquement pour chercher à effectuer des révisions minimales. Les treillis de généralisation et de spécialisation dans APT ([Nédellec, 1994]), ou l'arbre de spécialisation de ML-SMART ([Bergadano et Giordana, 1990]) sont des exemples de telles structures. [Wrobel, 1996] définit ainsi un certain nombre d'opérateurs minimaux de révision pour la programmation logique inductive. Il définit ces opérateurs minimaux comme "*essayant de garantir une perte minimale de pouvoir inférentiel pour les exemples courants et futurs*". Les opérateurs minimaux listés par [Wrobel, 1996] sont uniquement des opérateurs de spécialisation, c'est-à-dire qu'ils font en sorte qu'un exemple négatif ne soit plus couvert par la théorie. Un des opérateurs de révision minimale cité par Stefan Wrobel est l'opérateur MBR de MOBAL qui maintient une liste d'exceptions pour les clauses révisées ([Morik, Wrobel, Kietz et Emde, 1993]).

- Le second type de révision utilise des opérateurs qui ont été très simplement construits en se basant sur la structure de l'objet à corriger (la règle). Dans le système EITHER ([Ourston et Mooney, 1990], [Ourston et Mooney, 1994]) sont par exemple distingués quatre types d'opérateurs : ajouter un littéral dans une règle, retirer un littéral dans une règle, ajouter une règle, retirer une règle. Ce type d'opérateur n'est pas minimal et une révision minimale ne peut éventuellement être obtenue que par une combinaison de plusieurs de ces opérateurs. Par contre, ces opérateurs simples non minimaux offrent l'avantage de maintenir la BC plus compréhensible. La BC n'est pas complexifiée par l'ajout de notations spécifiques pour la révision.

La révision dans REVINOS appartient à la seconde catégorie de révision. Nos trois opérateurs ne sont pas minimaux. Dans le contexte d'utilisation de REVINOS, le but n'est pas d'apprendre exhaustivement une "définition de nodCible" (*Couv(nodCible)*), c'est d'abord d'intégrer correctement l'exemple courant. La généralisation à des exemples voisins n'est spécifiquement recherchée que pour tester si l'opérateur plus simple *modifieRègle* pourrait être appliqué à la place de l'opérateur plus spécifique *remplaceRègle*. Notre démarche générale d'acquisition incrémentale des connaissances justifie cette priorité de l'intégration de

<exCourant> sur l'apprentissage de <nodCible>. Nous aurions eu besoin d'effectuer des révisions minimales si l'on avait voulu chercher à délimiter précisément les contours de l'objet <nodCible>. Or, comme nous avons les moyens, grâce à la soumission d'exemples similaires aux "concepts essai" de MARVIN et aux exemples déterminants d'APT, de vérifier que <exCourant> est "couvert" sans que "d'autres exemples négatifs ne soient aussi couverts", pour reprendre la terminologie de l'apprentissage automatique, le recours à des opérateurs simples mais non-minimaux est tout à fait suffisant[10]. Les sauts inductifs effectués sont ainsi toujours contrôlés.

Par rapport à la terminologie classique des opérateurs utilisés en révision ([Ourston et Mooney, 1990], [Wrobel,1996],...), nous pouvons classer nos trois opérateurs de la manière suivante :

- Notre opérateur *créeRègle* correspond à l'opérateur d'*ajout* d'une règle ("addition"). Il effectue une généralisation.

- Notre opérateur *modifieRègle* correspond à l'application successive de l'opérateur de *retrait d'une règle* ("*deletion*"), qui effectue une spécialisation, puis de l'opérateur d'*ajout* d'une règle.

- Notre opérateur *remplaceRègle* correspond à l'application de l'opérateur d'*ajout d'un littéral* ("*antecedent addition*"), qui effectue une spécialisation, et de l'opérateur d'*ajout* d'une règle.

3.6.4 La notion de couverture

La notion de "parcours d'un graphe" telle que nous l'avons introduite, est un peu similaire à la notion d'**extension** utilisée par exemple en logique des défauts ([Reiter, 1980]) :

"(../...) [Dans les logiques non monotones], il peut exister plusieurs ensembles (appelés extensions), satisfaisables mais mutuellement contradictoires de formules inférables à partir d'une théorie munie de conventions"
[Haton, Bouzid, Charpillet, Haton, Lâasri, Lâasri , Marquis, Mondot et Napoli, 1991, p. 96]

Notre notion de "parcours d'un graphe" pourrait correspondre ainsi à l'ensemble des extensions possibles d'une théorie logique équivalente au graphe, à ceci près que dans notre approche, les différents parcours d'un graphe n'ont pas l'obligation d'être mutuellement contradictoires. Comme nous l'avons remarqué dans

[10] Cela se justifie moins lors de l'étape 4 de généralisation proprement dite, où l'on peut alors vraiment chercher à "apprendre <nodCible>". D'autres opérateurs seraient donc utiles.

la section 3.6.2, les seules différences qu'on peut attendre de deux parcours d'une même couverture sont des différences de nature conceptuelle : deux parcours distincts doivent être susceptibles de différencier deux situations concrètes éventuelles.

La notion de parcours dans REVINOS est sans doute plus proche de la notion d'**environnement** dans les systèmes dits ATMS ([De Kleer, 1986]). Un ATMS, pour "*Assumption-based Truth Maintenance System*", est un système de maintien de vérité :

> *"Les systèmes de maintien de vérité constituent une approche intéressante pour gérer des hypothèses. Un de leurs objectifs est en effet d'assurer la cohérence d'une base de connaissances en révisant continuellement les déductions sensibles à l'évolution de la base de faits"*
> [Haton, Bouzid, Charpillet, Haton, Lâasri, Lâasri , Marquis, Mondot et Napoli, 1991, p. 231]

Le rôle d'un ATMS est de détecter si les hypothèses contenues dans la base de faits sont incohérentes, pour ensuite tenter de réviser la BC en conséquence. Un environnement est défini par un ensemble d'hypothèses compatibles. L'ATMS construit un réseau de justifications dont chaque **nœud**, qui correspond à une formule atomique instanciée, comprend une **étiquette** (*label*). L'étiquette d'un nœud comprend l'ensemble minimal des environnements qui permettent de dériver le nœud. L'ATMS calcule et met à jour l'étiquette de chaque nœud. La tâche de l'ATMS est de maximiser un ensemble d'hypothèses cohérentes entre elles, c'est-à-dire de calculer l'environnement "maximal".

On peut ainsi faire l'**analogie entre nœud et nodule, et entre étiquette du nœud et couverture du nodule.** Notre notion de parcours ou d'exemple abstrait diffère sensiblement de la notion d'environnement dans un ATMS puisque les hypothèses contenues par un environnement sont équivalentes à des faits instanciés. Mais d'une manière générale, le calcul de la couverture dans REVINOS s'apparente au calcul de propagation que l'ATMS effectue pour mettre à jour les étiquettes.

L'approche ATMS a été utilisée dans deux systèmes de révision, RTLS ([Ginsberg, 1988c]) et STALKER ([Carbonara et Sleeman, 1996]), avec deux orientations complètement différentes.

En premier lieu, la représentation ATMS peut s'avérer très utile pour tester instantanément si un exemple donné peut dériver un nœud (une conclusion). Il suffit ainsi de tester si l'exemple est dans un des environnements de l'étiquette. Le

système STALKER est le successeur du système KRUST. Comme KRUST (voir section 3.2.2.4), STALKER gère simultanément un ensemble de corrections possibles dont il voudrait évaluer l'efficacité pour effectuer un choix définitif. STALKER construit et maintient un ATMS, dans lequel les environnements sont tout simplement les exemples positifs. Le système propage successivement chaque opération de révision candidate pour ensuite évaluer rapidement les exemples couverts. STALKER se sert donc d'un ATMS pour évaluer les répercussions des corrections, d'une manière similaire à REVINOS. Deux différences notables cependant peuvent être mentionnées, qui sont liées au fait que REVINOS soumet des exemples à l'utilisateur, contrairement à KRUST et STALKER qui ne sont pas interactifs :

- REVINOS calcule des exemples *abstraits* ,
- REVINOS calcule ces exemples à partir des *différences* entre les étiquettes avant et après l'application de la correction.

RTLS est un deuxième exemple d'application de l'approche ATMS à la révision. Contrairement à STALKER, RTLS s'appuie directement sur la structure ATMS pour réviser. RTLS intervient après que la théorie ait été "réduite" par un autre système, le bien-nommé KB-REDUCER ([Ginsberg, 1988b]) qui "opérationnalise" une base de règle en fabriquant une structure similaire à un ATMS. RTLS va donc directement réviser l'ATMS. Une troisième étape, décrite dans [Ginsberg, 1990] décrit l'algorithme inverse de retraduction en base de règles "non réduite".

Ce qui fait la particularité de RTLS, c'est que l'algorithme de révision travaille directement sur les environnements d'une étiquette. Autrement dit, le résultat de la révision est une nouvelle étiquette. Allen Ginsberg pose le problème de la généralisation de la manière suivante : il convient de généraliser quand un des environnements de l'étiquette d'une assertion n'est pas vérifié pour un exemple positif. RTLS va donc chercher à généraliser cette étiquette avec la stratégie suivante :

"Au cours de la résolution d'un problème de généralisation (spécialisation), soyez sûr de ne pas créer de nouveaux problèmes de spécialisation (généralisation)"

[Ginsberg, 1988c, p. 592]

L'algorithme de RTLS que nous allons décrire maintenant est appelé "généralisation focalisée d'étiquette" (*Focused Label Generalization*). Nous allons voir qu'il présente quelques points communs avec notre algorithme de révision.

RTLS dispose d'un ensemble d'exemples positifs et d'exemples négatifs pour l'assertion courante, d'un ensemble d'environnements qu'il faut généraliser, et d'un ensemble d'exemples mal classés par la théorie (que nous appellerons “exemples à apprendre”). L'opérateur qu'applique RTLS est l'opérateur d'abandon de littéraux.

RTLS commence par essayer de généraliser les environnements qui corrigent un exemple à apprendre, lorsqu'on leur retire un seul littéral. RTLS regarde alors si ces environnements notés e - 1 ne couvrent aucun exemple négatif. Si c'est le cas, les exemples à apprendre correspondants sont intégrés tout simplement en rajoutant e - 1 dans l'étiquette.

Lorsqu'il y a des environnements e - 1 qui couvrent des exemples négatifs, RTLS cherche tous les observables (faits) notés o qui seraient présents dans l'exemple à apprendre et absents dans ces exemples négatifs couverts par e - 1. S'il en trouve, les exemples à apprendre correspondants sont intégrés en ajoutant dans l'étiquette e - 1 + o (RTLS retire le littéral identifié auparavant et rajoute les observables o).

Si RTLS ne trouve pas d'observables de cette manière, il en cherche à partir de la liste des observables présents dans les exemples à apprendre, mais absents dans e - 1. S'il trouve ainsi une conjonction d'observables O qui ne couvrent aucun exemple négatif, les exemples à apprendre correspondant sont intégrés en rajoutant e - 1 + O dans l'étiquette.

S'il reste des exemples à corriger, RTLS reprend le même processus en essayant de retirer deux littéraux et donc en testant e - 2, e - 2 + o, etc. RTLS continue ainsi de suite à chercher à abandonner un nombre de plus en plus grand de littéraux.

Un algorithme similaire permet inversement de spécialiser les environnements d'une étiquette. Allen Ginsberg propose également un algorithme de “révision massive d'étiquette” pour restructurer globalement une étiquette à partir de méthodes statistiques, lorsque les “exemples à apprendre” de l'étiquette sont beaucoup plus nombreux que les exemples correctement classés.

On peut noter une certain ressemblance entre l'algorithme de “généralisation focalisée d'étiquette” de RTLS et notre algorithme de correction de graphe. Si l'on prend le vocabulaire de RTLS, on peut dire que REVINOS cherche au cours de l'étape 2 de révision à ajouter un environnement connu <exCourant> à <nodCible> et à le retirer de <ancienNod>. On cherche donc à spécialiser <ancienNod> pour généraliser <nodCible>. Il s'agit en quelque sorte d'un transfert d'environnement d'une étiquette à une autre. Ce problème ne se pose pas de la même manière dans RTLS où les étiquettes sont indépendantes les unes des autres : RTLS peut retirer un environnement d'une étiquette sans se soucier de ce que va devenir cet

environnement[11]. Mais on trouve le même problème de fond dans RTLS : chercher à généraliser une étiquette donnée pour qu'un exemple donné soit couvert.

REVINOS d'une manière générale dispose de plus de biais ou de contraintes que RTLS et cela aide à focaliser la révision, comme nous l'avons déjà remarqué dans la section 3.6.1. Par exemple, pour le choix du littéral à abandonner dans REVINOS, il est logique de se tourner d'abord vers le fait renvoyé par l'action de <nodDépart>. Si l'abandon de ce fait-là suffit, REVINOS applique *créeRègle* ou *modifieRègle* et cela correspond à une révision RTLS "e - 1".

Contrairement à RTLS, REVINOS peut aussi utiliser des connaissances explicites contenues dans les graphes de nodules, mais dont RTLS ne dispose pas : les règles de choix des nodules. Ainsi, lorsque l'opérateur *modifieRègle* est appliqué, le contenu des prémisses de la règle activée a fourni immédiatement à REVINOS la liste d'observables pertinents, que RTLS doit par contre rechercher "en aveugle".

Enfin lorsque REVINOS, après avoir généré des exemples que l'utilisateur a classé négatifs, sait qu'il doit utiliser l'opérateur *remplaceRègle*, l'objectif est le même que celui de RTLS : les observables que cherche RTLS correspondent aux faits discriminants que cherche REVINOS (section 3.3.2). Là encore la recherche de REVINOS est contrainte par les prémisses de la règle activée.

La notion de couverture d'un nodule présente également des similarités avec le fonctionnement du système ENIGME ([Thomas, Laublet et Ganascia, 1993], [Thomas, 1996]) qui intègre des outils d'apprentissage automatique dans une méthodologie d'acquisition des connaissances. ENIGME décompose l'apprentissage de la connaissance opérationnelle en plusieurs étapes, qui chacune correspond à un opérateur de la structure d'inférence KADS. Lorsque ENIGME apprend les connaissances opérationnelles d'un pas d'inférence donné, il utilise la structure d'inférence KADS pour restreindre le sous-ensemble des exemples d'apprentissage utilisés. Cette restriction est la conséquence de la même idée que la notion de couverture d'un nodule dans REVINOS : "l'historique" du raisonnement contraint l'ensemble des exemples concernés par une étape donnée du raisonnement.

3.7 Travaux futurs

[11] Dans l'approche d'A. Ginsberg, cela rendra d'ailleurs plus tard difficile l'étape de retraduction ([Ginsberg, 1990]) : cette dissociation entre révision et retraduction entraîne une perte d'information qui complique la tâche de retraduction.

Un certain nombre d'améliorations peuvent être apportées à REVINOS, à commencer par le dialogue durant l'étape 1. Rappelons que ce dialogue (et le graphe appelé "REVINOS") a été l'objet d'enrichissements constants depuis sa création.

• Ainsi, dans le cas d'une information manquante (section 3.2.1.2), l'action choisie ou créée par le concepteur pour renvoyer cette information manquante est pour l'instant ajoutée en fin de parcours. Il serait intéressant de proposer à l'utilisateur de choisir à quel niveau du parcours doit s'insérer le nouveau nodule.

• REVINOS gère implicitement la liste des exemples à réviser. Il serait très intéressant pour le concepteur d'avoir accès à cette liste afin qu'il puisse gérer lui-même l'incrémentalité de la révision : choisir l'exemple qu'il veut corriger à présent, annoter les autres, les sauvegarder, etc.

• Au cours de la phase de validation des exemples (étape 3), il paraît important que l'utilisateur puisse demander une décomposition de l'exemple abstrait affiché, lorsque celui-ci lui semble d'un niveau d'abstraction trop élevé pour pouvoir être accepté tel quel. C'est principalement dans le cas d'attributs valués que cette opération apparaît utile. Par exemple, dans le cas où l'exemple abstrait définit un intervalle pour une valeur, sur lequel l'utilisateur ne peut pas se prononcer, l'utilisateur peut vouloir découper cet intervalle en deux et se prononcer ensuite sur chacun des deux exemples abstraits

• Parmi les autres opérateurs de révision qui pourraient être intégrés à REVINOS, il en est un, inspiré du système KRUST, qui se contente de modifier les niveaux de priorité des règles. Cet opérateur consisterait à effectuer les opérations suivantes :

- repérer une éventuelle règle r2 activant <nodCible>, qui aurait pu être activée à la place de la règle r1 si elle avait eu une priorité supérieure à celle de r1.

- si r2 existe, placer r1 à un niveau de priorité inférieure à r2 et en calculer les répercussions,

- si les répercussions de la tentative précédente ne sont pas validées, recommencer en montant cette fois-ci le niveau de priorité de r2 au-dessus de celui de r1.

Notons que cet opérateur ne peut être "essayé" que dans un nombre réduit de situations (il faut la présence d'une règle candidate "masquée"). De plus, son efficacité est assez relative, contrairement à l'opérateur *remplaceRègle*, dont le résultat est assuré. Enfin la notion de priorité dans les règles de choix permet de stocker des connaissances qui n'ont pas encore été explicitées sous la forme de conditions dans des prémisses. Il est préférable de chercher à expliciter ces "observables" comme le permet *remplaceRègle*.

• A la suite de l'application d'un opérateur de révision, il serait intéressant de disposer d'une fonction qui “réharmonise” les règles de choix sans modifier la couverture du nodule. Il arrive en effet par exemple que certaines priorités n'aient plus aucun sens à la suite de l'application de plusieurs opérateurs de révision.

• L'opérateur *remplaceRègle* peut être rendu plus efficace si REVINOS propose au concepteur de valider les faits discriminants détectés, lorsque ceux-ci sont nombreux. Cela pourrait éventuellement éviter de construire une règle trop spécifique.

• La similitude avec les ATMS conduit à l'idée d'éviter de toujours recalculer les couvertures, en stockant la couverture de chaque nodule. Cela permettrait d'accélérer le calcul des répercussions, surtout lorsque des sous-graphes sont appelés.

3.8 Conclusion

L'outil REVINOS permet à un concepteur novice de corriger facilement des graphes, sans jamais visualiser de règles de choix. Les seuls objets que visualise, modifie et crée l'utilisateur / concepteur, sont les nodules, les actions et les faits. Il est d'ailleurs important que ces objets soient maintenus par le concepteur afin que la compréhensibilité de la BC soit préservée.

L'utilisation de REVINOS s'avère également très utile pour un “expert en construction de graphes de nodules” pour les raisons suivantes:

- REVINOS permet un suivi complet des répercussions des corrections effectués et gère les exemples à corriger;

- se laisser porter par le dialogue guidant la révision est confortable et en fin de compte efficace;

- le partage des tâches dans REVINOS est utile et le concepteur est heureux d'être déchargé de la maintenance des règles.

Ce qui fait l'originalité de REVINOS par rapport aux autres systèmes de révision, c'est le fait que la structure utile à la révision est précisément celle utilisée pour la BC elle-même. Les graphes de nodules n'ont pas besoin d'être transformés en arbres de preuve : ils servent tels quels de support au processus de révision

De plus, comme les nodules de situation ont pour caractéristique d'être compréhensibles, parce qu'ils sont les objets de base de la BC et parce qu'ils ont été commentés, le concepteur a l'opportunité de directement désigner ces objets au cours de la révision. Le problème de révision est alors extrêmement contraint par rapport aux systèmes de révision traditionnels qui ne peuvent pas faire explicitement

référence à leur structure de révision parce que celle-ci ne correspond pas à la BC connue de leur utilisateur.

La notion d'exemple abstrait est une seconde particularité de REVINOS. Ces exemples abstraits, qui ont pu être calculés grâce à la déclaration des actions, permettent d'améliorer la communication entre l'utilisateur et le système. Les exemples générés et soumis par REVINOS sont en effet moins nombreux et plus informatifs que les exemples concrets.

Chapitre 4 :

Quelques représentations similaires aux nodules de situation

Dans le chapitre 1, nous avons exposé comment nous avons abouti à la représentation des connaissances par nodules de situation à partir du modèle Situation-Action. Nous avons montré dans les chapitres 2 et 3 comment les nodules de situation étaient une réponse à l'objectif que nous nous étions assigné : faciliter la construction incrémentale de BC opératoires. Ce chapitre 4 fait un inventaire comparatif des formalismes qui présentent des ressemblances avec le nôtre.

En établissant les ressemblances et différences avec des modèles de représentation des connaissances proches des nodules de situation, il sera possible de bien cerner ce qui fait l'identité et l'originalité de notre modèle. D'autre part, déceler des points communs avec un grand nombre de formalismes plus ou moins bien connus peut être interprété comme un signe de pertinence de notre modèle : notre travail prend sa place au sein d'un ensemble de recherches qui partagent, ou bien des objectifs similaires, ou bien des intuitions communes.

Nous commencerons par présenter quatre représentations assez proches des nodules de situation, les Règles Dé-Roulées (Ripple-Down Rules) de Paul Compton, les Graphes d'Exceptions de Brian Gaines, les arbres de décision et les Réseaux de Transitions Enrichis. Pour comparer ces représentations entre elles et avec la nôtre, nous reprendrons l'exemple des lentilles de contact introduit par

[Cendrowska, 1987], puis repris par [Gaines et Compton, 1995] et [Gaines, 1996]. Cet exemple, donné dans la figure 4.1 sous la forme de règles simples, possède l'avantage d'avoir déjà été utilisé pour comparer les différents formalismes. Notons en passant que nous discuterons pas dans ce chapitre de la représentation sous forme de règles de production puisque nous avons déjà abordé ce point dans la section 1.2.1.1.

Nous présenterons ensuite quelques représentations issues des travaux sur la représentation de la notion de *contexte*, sur la représentation de la notion de *plan*, et enfin sur la représentation de la notion de *tâche*.

r1: SI **larmes** = réduit
ALORS **lentilles** = non
r2: SI **astigmatisme** = non ET **prescription** = myope ET **âge** = presbyopique
ALORS **lentilles** = non
r3: SI **astigmatisme** = oui ET **prescription** = hypermétrope ET **âge** = presbyopique
ALORS **lentilles** = non
r4: SI **astigmatisme** = oui ET **prescription** = hypermétrope ET **âge** = pré-presbyopique
ALORS **lentilles** = non
r5: SI **larmes** = normal ET **astigmatisme** = oui ET **prescription** = myope
ALORS **lentilles** = dures
r6: SI **larmes** = normal ET **astigmatisme** = non ET **prescription** = hypermétrope
ALORS **lentilles** = souples
r7: SI **larmes** = normal ET **astigmatisme** = non ET **âge** = pré-presbyopique
ALORS **lentilles** = souples
r8: SI **larmes** = normal ET **astigmatisme** = non ET **âge** = jeune
ALORS **lentilles** = souples
r9: SI **larmes** = normal ET **astigmatisme** = oui ET **âge** = jeune
ALORS **lentilles** = dures

Figure 4.1 : L'exemple des lentilles (représentation par règles)

4.1 Les Règles Dé-Roulées (RDR)

La représentation par Ripple-Down Rules, que nous proposons de traduire par Règles Dé-Roulées, traduction qui conserve les initiales “RDR”, est la représentation des connaissances que nous nous devions de présenter en premier. Introduite par [Compton et Jansen, 1990], cette représentation a été en effet définie de manière à répondre à des objectifs quasiment similaires aux nôtres. Pour [Compton et Jansen, 1990], qui se réfèrent notamment à Karl Popper et au mouvement “cognition située” ([Compton, 1992]), les justifications que donnent les experts à propos de leur raisonnement, sont toujours reconstruites après coup :

“La méthodologie d'acquisition des connaissances à l'aide de Règles Dé-Roulées est basée sur l'idée que la connaissance fournie par les experts n'est pas un exposé de comment ils ont atteint une conclusion, mais une justification en contexte du fait que leur jugement était en fin de compte correct”

[Kang et Compton, 1992, p. 140]

La connaissance (justification) donnée dans un contexte spécifique n'est valable et ne doit être ensuite utilisée que dans ce même contexte ([Compton, Preston, Kang et Yip, 1994]). Par conséquent l'approche "Règles Dé-Roulées" propose, plutôt qu'une acquisition des connaissances par modélisation préalable, une acquisition incrémentale des connaissances basée sur des **cas validés** ("*Validated Cased Based Knowledge Acquisition*") ([Compton, Preston, Edwards et Kang, 1996]).

[Compton et Jansen, 1990] donnent les conseils suivants pour la construction de SBCs :

- ne pas chercher à généraliser la connaissance au moment où on l'acquiert,
- essayer plutôt d'enregistrer le contexte dans lequel chaque connaissance est acquise,
- les BCs doivent être conçues pour évoluer.

[Compton et Jansen, 1990] argumentent pour que les révisions des BCs soient faites localement et prudemment :

"... on dot être capable de modifier n'importe quel aspect de la connaissance sans corrompre l'ensemble, ou du moins être en mesure de contrôler complètement les modifications"

[Compton et Jansen, 1990, p. 251]

La représentation par Règles Dé-Roulées consiste à l'origine ([Compton et Jansen, 1990]) à ajouter une contrainte explicite sur le fonctionnement du moteur d'inférence du système expert GARVAN-ES1. Chaque règle de la Base de Règles est complétée par un prédicat "DERNIÈRE_ACTIVÉE" (*LAST_FIRED*) contenant le numéro d'une autre règle qui doit avoir été activée pour que la règle courante soit elle-même activable. Une règle ne peut ainsi être activée que si son "ancêtre" l'a été. Chaque règle définit alors un "contexte" particulier qui correspond à la fois à l'ensemble des conditions vérifiées par toutes les règles qui ont été activées[1]. Le modèle a ensuite évolué avec la définition pour chaque règle R de deux règles successeurs éventuelles ([Compton, Edwards, Kang, Lazarus, Malor, Menzies, Preston, Srinivasan et Sammut, 1991]) : une règle sera définie pour être activable lorsque les conditions de R sont vérifiées (on parlera de lien "if-true"), une autre pourra être définie pour le cas où les conditions de R ne sont pas vérifiées (on parlera de lien "if-false").

Les Règles Dé-Roulées proposent donc un moyen :

---

[1] Cette notion de *contexte* dans les RDRs équivaut donc à la notion de *parcours* dans la représentation par nodules de situation.

- d'exprimer les relations entre les différentes règles d'une base de règles. Les règles d'une BC deviennent les nœuds d'un arbre binaire, parcouru à partir de sa racine jusqu'à un nœud / règle final(e).

- de stocker les cas résolus dans chaque nœud / règle concerné(e).

- de pouvoir introduire les nouvelles règles au niveau de généralité désiré. Un nouveau cas peut être intégré en ajoutant simplement une règle if-true ou if-false à la règle R qui était activée pour ce cas. La structure des RDRs fait que cet ajout n'a aucune répercussion sur les cas autres que ceux concernés par R.

La figure 4.2 donne la représentation par Règles Dé-Roulées de l'exemple des lentilles de contact, tel que donné par [Gaines et Compton, 1995]. La règle R1 ne contient aucune condition : elle est toujours déclenchée, et par conséquent le lien "if-true" est toujours suivi pour tenter d'activer la règle R2. La conclusion "lentilles = non" de la règle R1 est la conclusion par défaut, si aucune des autres conditions n'est vérifiée. La règle R4 équivaut ici à la règle r5 de la représentation par règles de la figure 4.1, tandis que la règle R2 équivaut aux règles r6, r7 et r8.

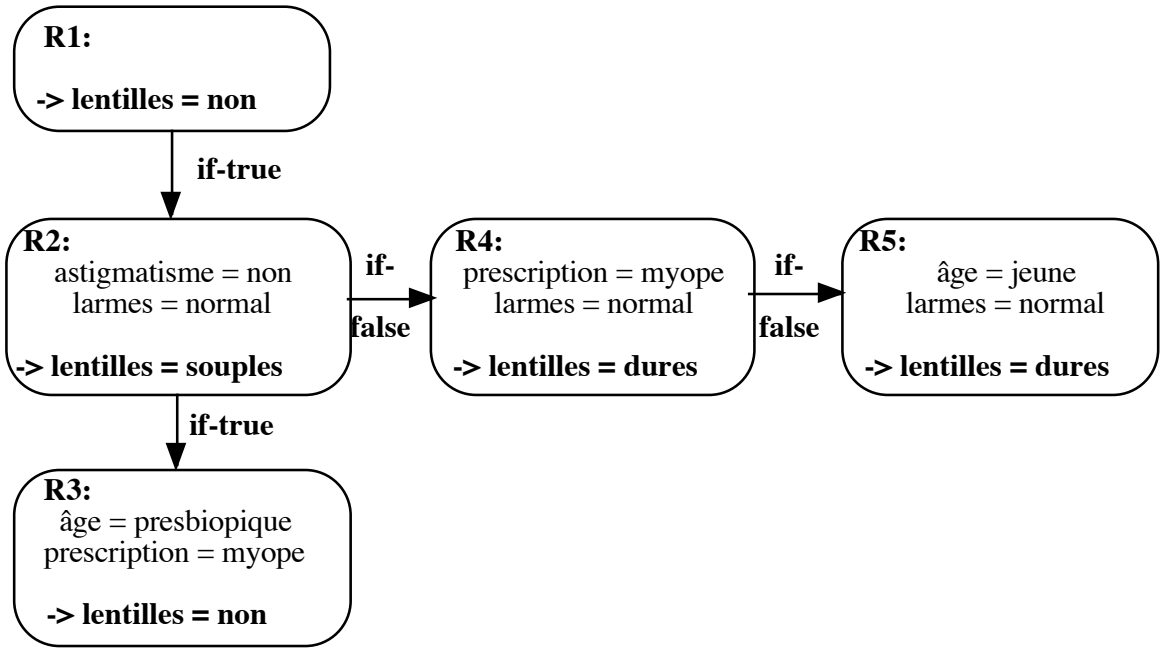


Figure 4.2 : L'exemple des lentilles (représentation par Règles Dé-Roulées)

En proposant une représentation des connaissances pour faciliter la maintenance et la construction incrémentale des BCs, l'approche de Paul Compton et de ses collaborateurs rejoint notre approche dite "révision située" (section 1.2.3.2).

Deux différences notables peuvent cependant être remarquées, en plus de l'absence de la notion d'action[2] dans les RDRs :

- Notre approche met l'accent sur le fait que la BC se doit d'être **compréhensible**, propriété qui n'est pas centrale dans l'approche RDR. Alors qu'un nodule de situation est un objet qui doit pouvoir être compris par le concepteur, notamment en examinant sa position dans le graphe de nodules, deux Règles Dé-Roulées parentes n'ont pas toujours de lien sémantique clair (juste un lien d'exception). Dans l'approche Règles Dé-Roulées, c'est uniquement le contexte associé à un nœud (l'ensemble des cas concernés par le nœud) qui a une signification pour le concepteur : le chemin dans les RDRs pour parvenir à ce nœud n'importe pas. La contrainte d'interruptibilité (cf. section 1.1.4) de notre approche nous oblige à ce que chaque étape du raisonnement (nodule) aient un sens.

- De manière très liée au premier point, nous accordons une grande importance à la **révision** de la BC, c'est-à-dire à la capacité de corriger des éléments de connaissances erronés. L'approche Règles Dé-Roulées procède toujours par extension de la BC, puisque les nouveaux cas particuliers sont intégrés en ajoutant une exception supplémentaire à une règle :

> *"Les règles sont ajoutées, jamais modifiées"*
>
> [Compton, Kang, Preston et Mulholland, 1993, p. 284]

Par contre, REVINOS propose toujours de réutiliser des nodules (voir l'étape 1 de la révision dans la section 3.2) et une révision peut consister en une simple correction de branche dans un graphe, sans création de nouveaux objets.

Ces deux aspects sont liés dans la mesure où si l'on cherche à toujours réutiliser des objets déjà définis avant d'en créer de nouveaux, on améliore la compréhensibilité globale de la BC. Un autre aspect qui aide à la concision de la BC, donc à sa compréhensibilité, est la possibilité d'appeler des sous-graphes par l'action "déclencher" (section 2.1.3.2.b). Enfin, en donnant une signification à nos nodules (chacun représente une situation qu'il est possible d'expliciter), les nœuds sont des objets davantage familiers, compréhensibles et facilement manipulables par le concepteur.

En résumé, on peut voir les Règles Dé-Roulées comme des graphes de nodules de situation où :

---

[2] La notion d'action est une particularité de notre représentation qu'on ne retrouvera dans aucun des modèles présentés dans ce chapitre. Rappelons que cette notion d'action permet au concepteur :
- d'aller chercher ou calculer uniquement les informations utiles au raisonnement courant et de préciser l'ordre dans lequel ces informations doivent être "pêchées",
- de réutiliser des parties de la BC (action type "déclencher") et de permettre ainsi une certaine modularité.

- les actions des nodules ne sont que des conclusions et ne renvoient pas de fait (tous les faits sont déjà stockés dans la MT avant l'activation du nodule initial).

- un nodule ne peut avoir que deux nodules successeurs et les règles de choix sont toutes de la forme : “si: <formule> alors Aller: <noduleIfTrue> priorité: 1; aller: <noduleIfFalse>”.

Cette dernière restriction n'apparaît plus dans les Règles Dé-Roulées pour la classification multiple. L'approche RDR a été en effet étendue pour permettre la classification multiple, c'est-à-dire permettre de conclure simultanément sur plusieurs nœuds qui correspondent à des classes différentes. Avec les MCRDR (Multiple Classification RDR), un nœud peut avoir plusieurs successeurs, ces derniers n'étant pas exclusifs ([Preston, Compton, Edwards et Kang, 1996], [Kang, Compton et Preston, 1998]). Autrement dit, plusieurs parcours simultanés dans l'arbre sont possibles, ce qui amène une différence importante avec les nodules de situation et les RDR simples, puisqu'il n'y plus de bijection entre nœud et cas (ou contexte).

4.2 Les Graphes d'Exceptions

[Gaines, 1996] propose une représentation qui partage avec les MCRDR la caractéristique d'offrir des branches multiples qui ne s'excluent pas mutuellement : les Graphes d'Exceptions Orientés Acycliques (en anglais *Exception Directed Acyclic Graphs* ou EDAG).

Brian Gaines introduit la notion d'exception à une règle R pour permettre au concepteur d'expliciter les conditions particulières qui excluent les cas attachés à la règle R ([Gaines, 1991]). Contrairement à l'approche RDR, Brian Gaines recherche une représentation compréhensible et se base sur la notion d'héritage pour proposer les Graphes d'Exceptions, qui peuvent être lus comme des ensembles de règles d'exceptions ou comme des arbres de décision généralisés :

- Un chemin dans un graphe part du nœud racine et se poursuit jusqu'à ce que la prémisse du nœud ne soit pas vérifiée ou qu'un nœud-feuille soit atteint.

- Pour chaque nœud, si la prémisse est vérifiée, la conclusion du nœud remplace la conclusion courante.

- Plusieurs chemins sont possibles simultanément.

L'exemple des lentilles de contact, représenté sous forme de Graphe d'Exceptions, est donné dans la figure 4.3.

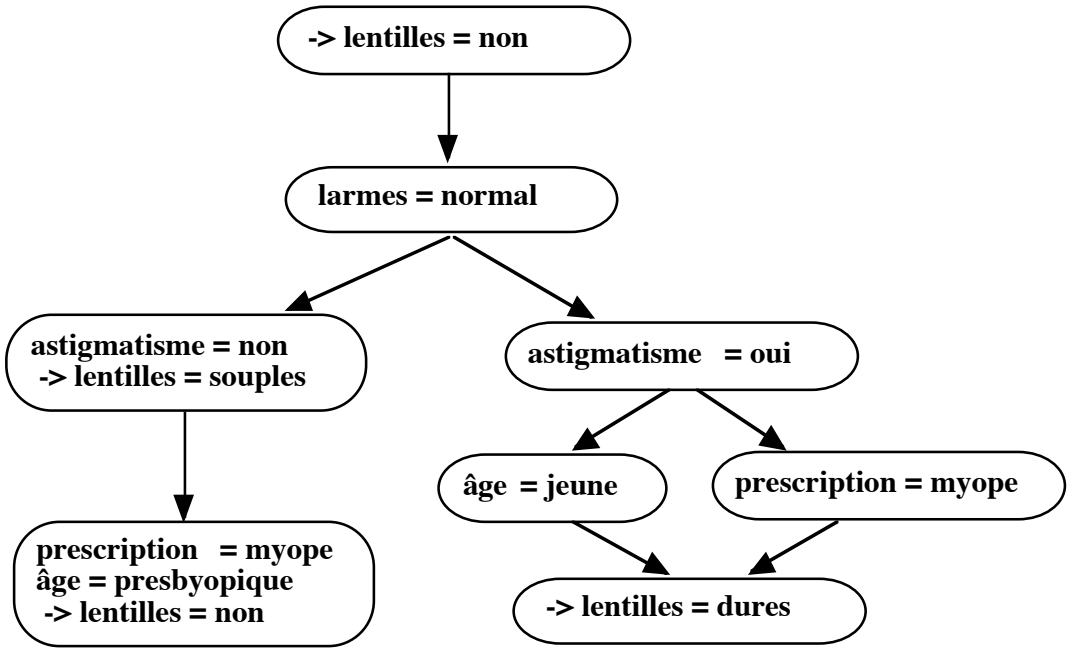


Figure 4.3 : L'exemple des lentilles (représentation par Graphe d'Exceptions)

Les liens entre deux nœuds peuvent être lus comme signifiants “sauf-si”, de manière similaire d'ailleurs aux liens “if-true” des RDR. Par exemple “si le patient n'est pas astigmate alors lui prescrire des lentilles souples, sauf si le patient est myope et d'âge presbyopique, où dans ce cas, il ne faut pas lui prescrire de lentilles”.

Chaque nœud d'un Graphe d'Exceptions peut avoir plusieurs successeurs et le graphe peut donc mener à plusieurs conclusions valides distinctes.

“D'un point de vue psychologique, on peut interpréter un Graphe d'Exceptions de la manière suivante : chaque nœud fournit un ‘contexte’ bien-défini pour sa conclusion, chaque nœud apparaît comme une exception à ses nœuds parents dans le graphe, et il a pour exception les nœuds immédiatement en dessous. Cette notion de contexte est celle suggérée par [Compton et Jansen, 1990] à la différence que les contextes des Graphes d'Exceptions ne sont pas nécessairement exclusifs”

[Gaines, 1996]

On peut remarquer qu'un Graphe d'Exceptions est une structure compacte et lisible, faisant apparaître bien clairement les conditions des nœuds (plus lisiblement par exemple que dans le graphe de nodules de la figure 4.4).

Toutefois deux critiques peuvent être formulées à propos des Graphes d'Exceptions :

- pour “faire tourner un cas” sur un Graphe d'Exceptions, il faut tester les conditions de toutes les sous-branches : comme pour un cas donné plusieurs nœuds différents peuvent être simultanément activés, la lecture d'un Graphe d'Exceptions par le concepteur est sans doute une opération plus complexe que la lecture d'un RDR et d'un graphe de nodules.

- la seconde critique s'applique aussi aux RDRs. Si l'on dit qu'un nœud correspond à un contexte (ensemble de cas), le parcours d'un Graphe d'Exceptions ou de RDRs demande toujours d'explorer les autres contextes fils, pour tester leurs conditions, avant d'éventuellement revenir au contexte père. Ce retour-arrière virtuel, qui n'est pas très naturel, est évité dans les graphes de nodules puisque les conditions d'activation d'un contexte / nodule sont exprimées au niveau du nœud père. C'est ce qui permet dans notre représentation à un nœud d'avoir un grand nombre de nœuds successeurs qui s'excluent mutuellement. De plus, il y a un fort couplage entre le nœud / contexte (le nodule) et la conclusion (l'action) : l'action attachée au nodule courant est toujours exécutée, tandis que dans un Graphe d'Exceptions, tant que le parcours et les tests des conditions des sous-branches n'est pas terminé, on ne sait pas si l'action du nœud courant sera exécutée ou non. Ici encore, les contraintes que nous nous sommes fixées dans la section 1.1, comme l'interruptibilité et la séquentialité du raisonnement, rendent notre représentation plus simple et plus compréhensible.

La figure 4.4 montre un graphe de nodules construit pour l'exemple des lentilles de contact. Rappelons pour mémoire que si la priorité d'une règle de choix n'est pas précisée, elle est la plus faible de toutes les autres règles.

Nous ferons deux commentaires à cet exemple :

- Le nodule “demandeAutresInfos” peut être “éclaté” de manière par exemple, dans le cas d'un patient astigmate pour lequel on voudrait savoir s'il est possible de lui donner des lentilles dures, à préciser si l'on souhaite demander d'abord l'âge ou d'abord la prescription (dans les deux cas, on peut éviter de poser une question).

- On peut aussi traiter le problème de la conclusion par défaut “pasDeLentille” en créant deux graphes : un sous-graphe qui identifie les cas où l'on peut donner des lentilles, un autre qui appelle le précédent et donne la conclusion par défaut “pasDeLentille” si aucune conclusion n'a été donnée précédemment. Lorsque le nombre de cas est important, ce procédé permet de conserver une bonne lisibilité pour le graphe appelé, dans lequel il n'est plus nécessaire de spécifier les cas où la conclusion par défaut s'applique.

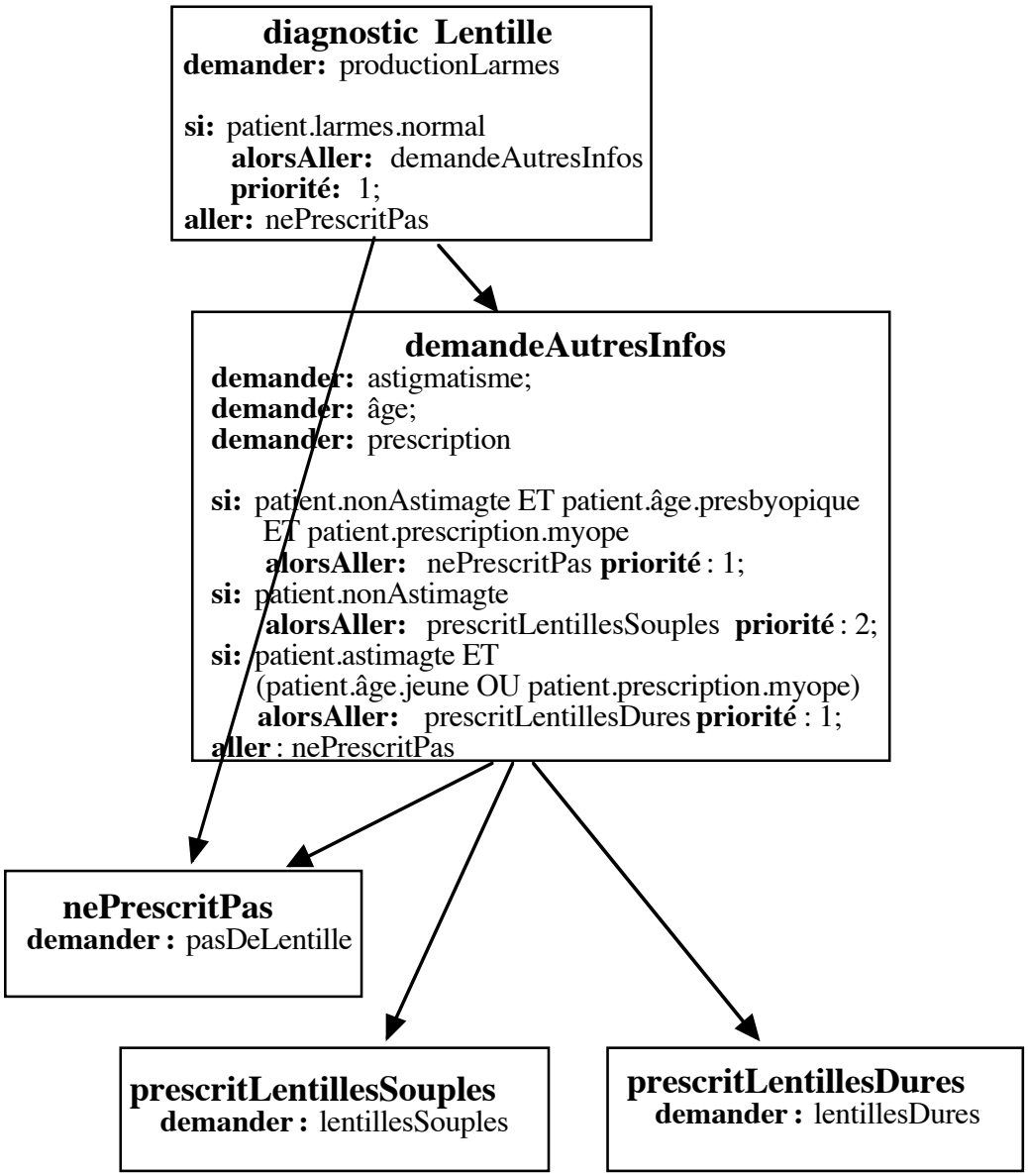


Figure 4.4 : L'exemple des lentilles (représentation par nodules de situation)

4.3 La famille des arbres de décisions

Les arbres de décision, tels que l'algorithme ID3 ([Quinlan, 1986]) les construit, peuvent être vus comme des graphes de nodules dont :

- chaque action est une simple “lecture” de la valeur d'un attribut, si l'on fait l'hypothèse que le système va chercher la valeur des attributs au fur et à mesure du parcours de l'arbre.

- chaque règle de choix ne peut comprendre dans ses prémisses qu'un test sur la valeur de l'attribut renvoyé par l'action courante.

- aucun nodule ne peut avoir de descendant commun : la structure est arborescente[3].

La figure 4.5 montre un arbre de décision pour l'exemple des lentilles de contact. Comparativement aux représentations déjà présentées, l'arbre de décision a pour caractéristique d'être la représentation la moins compacte, mais aussi celle pour laquelle l'application d'un exemple est le plus facile, puisqu'il suffit de suivre un chemin unique dans l'arbre qui explicitement "fléché" pour cela.

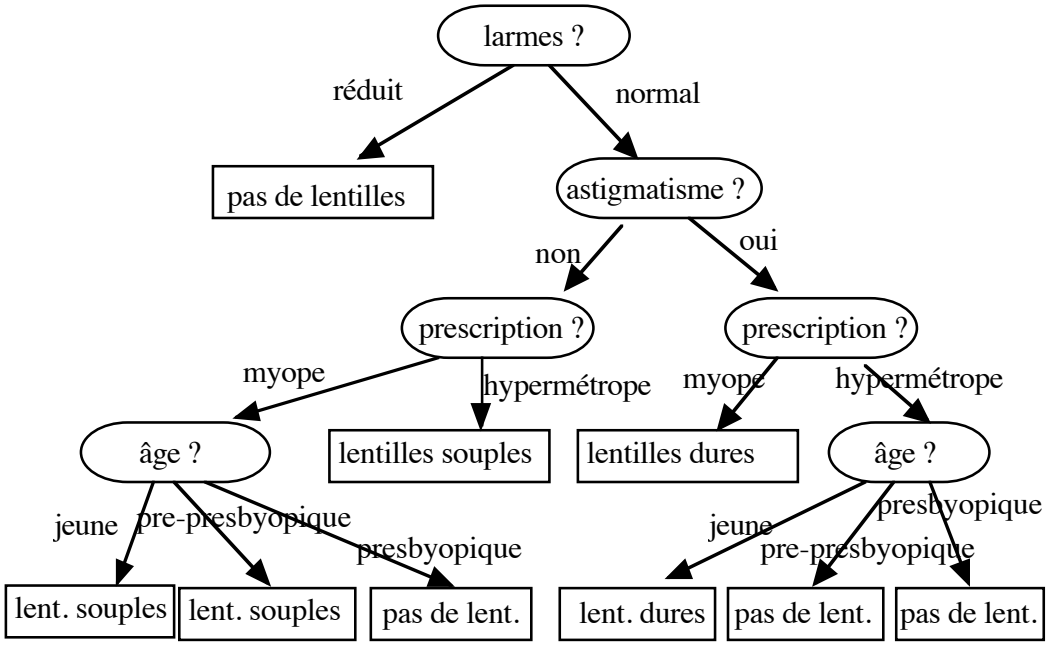


Figure 4.5 : L'exemple des lentilles (représentation par arbre de décision)

L'inconvénient de la représentation par arbres de décision est donc la taille des arbres et l'absence de modularité. Ces deux défauts n'apparaissent pas dans la représentation par nodules de situation, qui partage avec les arbres de décision la particularité de représenter explicitement les différents chemins possibles (fils de raisonnement). Les neuf règles de la figure 4.1 correspondent aux neuf chemins possibles dans l'arbre de la figure 4.5, mais la représentation sous forme de règles isolées ne permet pas d'exprimer ce fil du raisonnement qu'on veut explicite pour pouvoir facilement comprendre le système lorsqu'on l'interrompt. Les deux types de représentation peuvent être vues comme complémentaires : l'arbre de décision ou le

[3] Lorsque cette dernière condition n'est pas remplie, on parle de graphe de décision ou graphe d'induction ([Lebbe et Vignes, 1997]).

graphe de nodules explicite le fil du raisonnement, tandis que les règles indépendantes ou la couverture d'un graphe de nodules expriment les conclusions données pour chaque "contexte", au sens donné dans les RDRs et les Graphes d'Exceptions, d'*ensemble de cas*.

La représentation par nodules de situation permet d'obtenir une structure plus concise et plus lisible, puisque plusieurs branches de l'arbre de décision peuvent alors être regroupées au sein d'un même nœud (les règles des nodules peuvent porter simultanément sur les valeurs de plusieurs attributs). Dans le processus de transformation d'un arbre de décision en graphe de nodules plus compact, les parties communes d'un arbre de décision peuvent même donner lieu à la création d'un sous-graphe.

Un certain nombre de variantes ou de généralisations des arbres de décision ont été proposées ([Breslow et Aha, 1997]). Par exemple, dans les arbres de décision enrichis de [Cheng, Fayyad, Irani et Qian, 1988] un même nœud peut correspondre à plusieurs valeurs d'attribut. Citons également les "Oblivious Read-Once Decision Graphs" introduits par Ron Kohavi, qui ont la caractéristique de ne pas pouvoir comporter de boucles. Dans un "Oblivious Read-Once Decision Graph" ([Kohavi, 1993]) :

- chaque attribut ne peut être lu qu'une seule fois au cours d'un parcours,
- tous les nœuds d'un niveau donné ont pour successeur les nœuds d'un même autre niveau,
- tous les nœuds d'un niveau donné testent le même attribut.

Enfin mentionnons les listes de décision ([Rivest, 1987]) qui se situent quelque part entre les arbres de décision et les Règles Dé-Roulées. Une liste de décision peut être vue comme une liste de conditions chaînées par des liens "if-false". Par exemple, la liste "(condition1, conclusion1), (condition2, conclusion2), (vrai, conclusion3)" se lit de la manière suivante : si condition1 alors conclusion1, sinon, si condition2 alors conclusion2, sinon conclusion3. Les listes de décision sont finalement des ensembles linéairement ordonnés de règles. Les listes de décision peuvent également être vues comme des Règles Dé-Roulées dont les nœuds "if-true" ne peuvent comporter que des conclusions.

Par rapport aux arbres de décision, les listes de décision offrent une représentation plus générale et plus compacte puisque les conditions peuvent porter sur les valeurs de plusieurs attributs. Les listes de décision sont aussi un moyen d'introduire un fil de raisonnement dans un ensemble de règles. Mais ce fil, similaire

à celui des RDRs (donc n'ayant pas toujours de sens) est encore plus limité puisqu'il n'est pas possible de faire de bifurcations, juste des interruptions.

4.4 Les Réseaux de Transitions Enrichis

Le fonctionnement des graphes de nodules présente de grandes similarités avec celui des Réseaux de Transitions Enrichis (*Augmented Transition Networks* ou ATN : [Bobrow et Fraser, 1969], [Woods, 1970]). Les Réseaux de Transitions Enrichis sont des automates à états finis, augmentés de conditions locales de transition. Les transitions dépendent de la valeur de paramètres définis dans des registres globaux. Par exemple les registres des "Planning ATN" définis par [Miller et Goldstein, 1977] contiennent la description du problème, la solution retenue et des variables temporaires.

Les Réseaux de Transitions Récursifs ([Sabah, 1990]) permettent l'utilisation de sous-réseaux comme unités de transition : les règles de franchissement d'un arc peuvent alors être plus complexes.

Les graphes de nodules ressemblent beaucoup aux réseaux de transition si l'on fait la correspondance entre Mémoire de Travail et registres, entre déclenchement de sous-graphes et appel de sous-réseaux. Mais, comme les arbres de décision, les graphes de nodules n'acceptent pas de retour en arrière, alors que les réseaux de transition, qui sont utilisés pour l'analyse grammaticale et le traitement du langage naturel, s'en servent intensivement. Rappelons que notre objectif de prescriptivité exige que le concepteur précise toujours la démarche à suivre pour chaque situation prévue : cette approche qu'on a appelée "agir puis réviser" dans la section 1.3.6 ne nécessite pas de "recherche exploratoire"[4].

La notion d'action exécutée depuis les nodules est une particularité de notre formalisme qui le distingue des représentations décrites jusqu'à présent, pour lesquelles il est nécessaire de connaître au départ toutes les informations (valeurs d'attributs par exemples) qui seront utiles par la suite. Cette notion d'action est similaire à la notion de procédure dans les "réseaux syntaxicaux-sémantiques à nœuds procéduraux" ([Pierrel, 1987]). Les nœuds de ces réseaux peuvent appeler des procédures dont le rôle est de donner une "hiérarchie pour la prise en compte des sorties possibles du nœud" (donc pour le choix du nœud suivant : une procédure contient en plus presque l'équivalent de nos règles de choix...).

[4] Nous avons toutefois évoqué cette extension possible à notre modèle dans la section 1.3.6.

La figure 4.6 récapitule les propriétés de chaque formalisme que nous avons présentés jusqu'ici.

	Un nœud contient :	Nombre d'enfants possibles :	Nombre de parents possibles :	Plusieurs chemins en parallèle :	Appel d'une sous-partie :
Arbre de décision	test simple ou conclusion	plusieurs	1	NON	NON
Règles Dé-Roulées	prémisses ou conclusion	2 (plusieurs pour MCRDR)	1 (plusieurs pour MCRDR)	NON (OUI pour MCRDR)	NON
Graphe d'Exception	- prémisses - conclusion	plusieurs	plusieurs	OUI	NON
Réseau de Transition Enrichi	-test simple - test sur des états de registres	plusieurs	plusieurs	retour arrière et parallélisme peuvent être utilisés	NON (OUI pour les réseaux récursifs)
Graphe de nodules de situation	- actions - règles de choix	plusieurs	plusieurs	NON	OUI

Figure 4.6 : Tableau comparatif des représentations

Les travaux autour la notion de contexte ont conduit au développement de formalismes variés de représentations des connaissances, parmi lesquels on peut identifier des modèles similaires aux nodules de situation.

4.5 Les représentations des contextes

"Les contextes permettent de définir quelles connaissances devraient être considérées, quelles sont leurs conditions d'activation et limites de validité, ainsi qu'à quel moment elles doivent être utilisées"

[Brézillon, 1996, p. 6]

Les travaux en IA sur le contexte ([Brézillon, 1996], [Brézillon et Cavalcanti, 1997]) rassemblent différentes approches, qui, toutes, s'efforcent de prendre en compte dans la construction de SBCs la nature contextuelle de la connaissance. La notion de contexte intervient de deux manières en IA ([Brézillon et Abu-Hakima, 1995]) :

- pour aider à gérer correctement les grains de connaissances de la BC,

- pour aider à gérer la communication entre système et utilisateur.

C'est le premier type de travaux qui nous intéresse ici, puisqu'ils aboutissent souvent à la définition de représentations originales, parfois très proches de la nôtre. On peut ainsi remarquer un ensemble de travaux dont l'idée est de renoncer au concept d'une Base de Règle uniforme pour découper la base de règles en différents paquets indépendants, souvent appelés "contextes"[5], qui sont activés et désactivés soit, par un module "supérieur", soit directement par les règles d'un paquet. Le système SEPT ([Brézillon, 1990]) est un bon exemple de ce type d'architecture :

"A tout moment du raisonnement, le système considère uniquement les règles pertinentes de l'étape courante de la résolution de problème, et par conséquent le raisonnement du système se fait localement'

[Brézillon, 1996, p. 4]

Le système CENTAUR ([Aikins, 1983]) est précurseur de cette lignée de travaux qui attachent les règles à des structures, ce qui a pour effet de limiter le déclenchement et la portée des règles. CENTAUR utilise une combinaison de frames et de règles de production pour faire du diagnostic médical. Les frames sont appelées "prototypes", parce qu'elles représentent des situations typiques. Des règles sont attachées à certaines "facettes" des prototypes. Les connaissances de contrôle sont ainsi représentées à l'intérieur de chaque prototype ("contrôle spécifique au contexte"). De cette manière :

"L'expert peut spécifier 'ce qu'il faut faire' dans un contexte donné"

[Aikins, 1983, p. 165]

[Fikes et Keller, 1985] proposent également d'intégrer raisonnement basé sur les frames et raisonnement basé sur les règles :

"Une représentation basée sur les frames peut fournir une aide significative à la tâche de gestion des règles, en donnant un moyen d'organiser et indexer des collections modulaires de règles de production en fonction de leur usage prévu"

[Fikes et Keller, 1985, p. 916]

Dans [Fikes et Keller, 1985], une tâche de classification est accomplie en descendant dans une hiérarchie de prototypes. Mais en général, les prototypes contiennent les connaissances pour savoir si l'élément n'appartient pas à la classe, et

[5] Dans MYCIN les règles étaient déjà regroupées en "contextes" ([Buchanan et Shortliffe, 1984]), en fonction de la nature des conclusion des règles.

non pour savoir s'il lui appartient. Des règles associées aux prototypes sont donc créées pour exprimer des conditions d'appartenance à la classe.

Le système DIVA s'appuie aussi sur une hiérarchie de prototypes pour effectuer une tâche de reconnaissance de situation, vue comme une tâche de classification heuristique ([Krivine et David, 1988]). Un prototype correspond à une situation :

"Ces situations types sont organisées en hiérarchie, par des liens de raffinement; au plus haut niveau se trouvent les situations les plus générales, au plus bas niveau des situations très précises, auxquelles des diagnostics peuvent en général être directement attachés. (../...)
"Un prototype représente une situation type; un prototype est également un spécialiste de cette situation, et à ce titre sait comment se comporter"

[Krivine et David, 1988, p. 50]

Chaque situation-prototype contient des règles de diagnostic, qui permettent au prototype de se reconnaître ou de se rejeter, et des règles d'évocation pour activer d'autres prototypes. DIVA donne donc finalement une valeur numérique pour caractériser la valeur de confiance de la situation reconnue. Ce type d'information est utile lorsque le système a pour fonction de faire de la reconnaissance de situation ou de plan. Même si la reconnaissance de situation est au cœur de notre modèle, notre approche est différente parce que notre BC contient des connaissances prescriptives explicites : notre moteur doit exécuter l'action finale, donc doit lui-même faire des choix de manière explicite et définitive (définitive... tant que le concepteur n'exige pas une révision).

Nous terminerons cette section en présentant succinctement trois travaux dans trois domaines différents, qui ont pour point commun de s'appuyer sur une représentation modulaire des "contextes". Il s'agit :

- des *contextes* du système de diagnostic de pannes MOLTKE ([Althoff, Maurer et Rehbold, 1990], [Maurer, 1992]).

- des *modules contextuels* de l'architecture cognitive SimulACRUM ([Grant, 1994]).

- des *schémas contextuels et procéduraux* du projet ORCA consacré à la conception d'agents intelligents ([Turner, 1993]).

"Les contextes sont un des moyens de la modularité dans MOLTKE. Un contexte représente un diagnostic approximatif, intermédiaire ou final"

[Maurer, 1992, p. 339]

Un contexte dans MOLTKE ([Maurer, 1992]) est un objet qui comprend un nom, une précondition, une action de correction (MOLTKE fait du diagnostic), un ensemble de règles ordonnées qui prescrivent localement des stratégies de test et un ensemble de "règles de raccourci" (*shortcut rules*) capables d'inférer de nouveaux faits.

Les contextes sont organisés en graphe de contextes, dont les arcs signifient "est-un-raffinement-de". MOLTKE descend dans le graphe de contexte en commençant par le contexte initial et active un contexte lorsque ses préconditions sont remplies. La maintenance de la BC par le concepteur est assistée : MOLTKE maintient un réseau de dépendance des connaissances, qui précise toutes les relations entre contextes, règles, etc. Le système effectue des vérifications lors de l'insertion et du retrait des objets de la BC. Il cherche également à détecter des incohérences éventuelles.

Ces graphes de contextes sont donc des sortes de Graphes d'Exceptions enrichis, puisque les nœuds comprennent des actions (tests et inférences) en plus des conclusions (corrections). La différence notable avec les nodules de situation est dans la présence de préconditions pour chaque contexte. Nous avons déjà critiqué l'attachement de préconditions aux nœuds des Graphes d'Exceptions : nous reviendrons sur ce point dans la section 4.6.3.

'"Chaque contexte possède des règles spécifiques déclenchant les décisions ou les actions, une représentation cognitive spécifique des variables pertinentes dans le contexte, et des sources d'information spécifiques qui sont utilisées dans la dérivation des variables pertinentes"

[Grant, 1994]

Les modules contextuels de Simon Grant ont une visée générale de modélisation cognitive. Ces unités modulaires de petite taille correspondent aux différentes étapes de l'accomplissement d'une tâche.

"L'essence de l'approche contextuellement modulaire est que les régularités qui s'appliquent à certains contextes sont stockées ensemble, et sont accessibles ensemble. Lorsque des décisions ou des actions sont effectuées, les règles pour cela seraient inclues dans le module. De là, il n'y a qu'un pas pour y inclure les régularités qui déclenchent les transitions entre modules. Il pourrait y avoir, par exemple, des règles de la même forme que les règles de décision"

[Grant, 1994]

Deux types de transitions sont distinguées : les transitions "apprises", que S. Grant conçoit sous la forme d'un réseau de transition, et les transitions "générales" qui sont elles de nature associative.

La généralité du modèle de S. Grant rend difficile la comparaison avec notre modèle opérationnel, même si l'on peut déceler des intuitions communes.

"[Le] 'schéma contextuel' courant est alors utilisé pour guider virtuellement tous les aspects du comportement de l'agent, en assurant que le comportement soit automatiquement façonné à la situation. Quand la situation change, le schéma contextuel courant est mis à jour en conséquence, gardant automatiquement un comportement approprié pour sa situation de résolution de problème en évolution"

[Turner, 1993, p. 152]

Les schémas contextuels (c-schemas) de Roy Turner n'ont pas pour rôle de proposer les actions pertinentes pour la situation courante : ce rôle revient dans le projet ORCA aux "schémas procéduraux" (p-schemas). Les c-schemas sont d'un niveau conceptuel plus élevé : ils contiennent les buts qui peuvent être recherchés pour le contexte courant, les connaissances pour choisir un but selon la situation courante (l'activation des p-schemas permet d'atteindre ces buts), des connaissances sur la marche à suivre en cas d'événements inattendus, etc. Les deux types de schémas sont organisés en hiérarchie de généralisation / spécialisation. Un processus de remémoration à la CYRUS (voir section 1.2.1.3.b) permet à partir d'indices attachés aux c-schemas de choisir le c-schema courant. Rappelons que les hiérarchies d'épisodes généralisés de CYRUS sont assez proches des graphes de nodules : la différence est que dans notre modèle, les liens entre les nœuds sont explicités sous la forme de règles locales.

Toutefois, les graphes de nodules sont conceptuellement plus proches des p-schemas que des c-schemas de l'architecture ORCA. Mais la représentation explicite des contextes dans ORCA nous donne l'occasion de différencier les notions de contextes et de situations : un contexte est un objet plus persistant qu'une situation. Lorsqu'il est établit, l'objet "contexte" amène avec lui un socle de connaissances générales ou "par défaut" essentielles quand le système est un agent autonome opérant dans un environnement changeant.

Dans la section suivante nous allons présenter d'autres modèles proposés en robotique davantage similaires à nos graphes de nodules.

4.6 Les représentations des plans

4.6.1 La robotique située

Les robots situés sont des agents qui ont "une interaction complexe, permanente avec des environnements dynamiques et difficilement prévisibles"

([Rosenchein et Kaelbling, 1995]). Dans ce contexte, les concepteurs de robots situés ne peuvent pas trop s'appuyer sur l'équation "choisir un opérateur = choisir un état du monde" : l'environnement est constamment susceptible de changer et les post-conditions de l'opérateur choisi ne sont pas toujours vérifiées après l'exécution de l'opérateur (c'est ce qu'on appelle le *frame problem*). Par exemple les tables triangulaires ([Fikes, Hart et Nilsson, 1972]) fournissent une technique pour vérifier si le but courant est toujours accessible par le plan courant. Mais dans le cas contraire, il faut reconstruire complètement le plan...

C'est là un problème primordial pour la robotique située où il faut être capable de donner toujours et rapidement une réponse appropriée dans des environnements changeants. Une solution sans doute extrême consiste à renoncer à la notion de plan, et même à la notion de représentation explicite : l'architecture de subsomption ([Brooks, 1991]) consiste à construire des modules (automates à états finis) parallèles et indépendants, capables de s'activer et de s'inhiber les uns les autres.

Pour éviter d'avoir à effectuer les (re)constructions de plans, qu'elles soient faites à l'avance ou en situation, Michaël Georgeff et ses collaborateurs proposent une approche basée sur la description explicite de procédures, définies pour atteindre des buts donnés. [Georgeff et Bonollo, 1983] utilisent pour construire des "Systèmes Experts Procéduraux" des modules de connaissances (*Knowledge Areas*), constitués chacun d'un ensemble de règles d'invocation et d'un Réseau de Transitions Enrichi. Dans [Georgeff, Lansky et Bessiere, 1985], chaque procédure est représentée sous la forme d'un réseau de transition étiqueté, dans lequel le passage d'un nœud au suivant se fait si 1) la condition associée au nœud est vérifiée et 2) le but qui "étiquette" l'arc est rempli. Cette représentation est utilisée par un module capable de choisir la procédure adaptée pour le but recherché.

Dans la lignée de ces travaux, Marcel Schoppers introduit la notion de Plan Universel ([Schoppers, 1987], [Schoppers, 1995]) :

> *"… Les travaux sur les Systèmes Experts Procéduraux ne sont pas allés assez loin dans leur tentative de réaliser la réactivité : ils traitaient de la sélection (liée à la situation) de moyens pour réaliser des buts, mais pas de l'adoption et de l'abandon (liés à la situation) des buts"*
>
> [Schoppers, 1987, p. 1045]

L'approche de Marcel Schoppers est très similaire à la nôtre. Notre modèle Situation-Action, dans lequel nous allions une composante projet / tâche / but à une composante état du monde, rejoint le problème du roboticien qui doit allier réactivité et "finalité" (ou rationalité) :

"La planification est la sélection dirigée par les buts de réactions à des situations possibles. Cette approche fait le pont entre la planification dirigée par le but et la réaction dirigée par la situation[6]. Cela rend tout comportement réactif, mais permet la rationalité dans la sélection antérieure des réactions"

[Schoppers, 1987, p. 1041]

Les Plans Universel de M. Schoppers sont un moyen de recenser à l'avance les situations significatives qui peuvent se présenter au robot :

"... le planificateur doit anticiper les situations possibles et prédéterminer ses réactions dans ces situations"

[Schoppers, 1987, p. 1040]

Un Plan Universel ne contient pas de séquence d'action, en revanche :

"... la tâche du planificateur est de partitionner l'ensemble des situations possibles sur la base de la réaction que chaque situation requiert. Au moment de l'exécution, la situation actuelle est classée, et la réponse planifiée pour la classe de situation est alors effectuée"

[Schoppers, 1987, p. 1040]

Un Plan Universel se construit à partir des pré- et post- conditions des actions. Il indique, pour chaque but, les actions qui permettent d'atteindre ce but, en précisant les conditions nécessaires à vérifier ou les sous-buts à remplir. Un Plan Universel peut être ramené à un arbre de décision spécifiant l'action appropriée pour chaque situation. Cette action appropriée est exécutée tant que les conditions spécifiées par le Plan Universel restent vérifiées. Dès que celles-ci sont changées, le Plan Universel est de nouveau exploité et une nouvelle action exécutée.

Même si structurellement un Plan Universel ne ressemble pas à un graphe de nodules, on peut trouver beaucoup de points communs d'un point de vue conceptuel. Par exemple à propos de l'appel de sous-graphes ou de sous-plans :

"Ce qui est vu comme une 'action' par le plan x peut en fait être un autre Plan Universel"

[Schoppers, 1987, p. 1040]

L'expression du but d'un Plan Universel n'est pas aussi précise qu'elle l'est habituellement en planification :

"Le problème à résoudre est posé au planificateur comme une condition à réaliser, non comme un état du monde particulier"

[Schoppers, 1987, p. 1042]

---

6 Le terme *situation* chez Marcel Schoppers a le sens classique de "état du monde".

Notre représentation permet d'aller encore plus loin sur ce point, puisqu'un nodule initial n'est pas défini par une condition, mais il représente plus abstraitement une “attente” (cf. section 1.3.4) qui correspond à l'ensemble des nodules terminaux du graphe (ou à la “couverture” du graphe).

En résumé, la similarité entre Plans universels et graphes de nodules n'est pas surprenante puisqu'un graphe de nodules est un plan (il représente comment réaliser une certaine tâche) qui intègre les différentes alternatives qui peuvent s'ouvrir au cours de la réalisation de ce plan : c'est la définition même d'un Plan Universel.

L'objectif des travaux en reconnaissance de situations (dont la surveillance de processus), ou plus généralement, en reconnaissance de plans, est d'identifier une situation ou un plan à partir de l'observation d'un environnement changeant dans le temps. C'est là un objectif différent du nôtre, mais les objets utilisés pour la représentation des connaissances présentent des similarités fortes avec notre modèle.

4.6.2 La reconnaissance de plans

Alors que la planification consiste à proposer un plan à partir d'un ensemble de buts à remplir et d'actions pouvant être exécutées sur l'environnement, la reconnaissance de plan est le processus inverse qui consiste à trouver, à partir d'un ensemble d'actions effectuées par un agent :

- un ensemble de buts possiblement poursuivis par cet agent,
- un ensemble de plans typiques pour réaliser ces buts ([Cañamero, 1994]).

[Cañamero, Delannoy et Kodratoff, 1993] proposent pour la reconnaissance de plans une structure appelée XPlan, inspirée des travaux de Roger Schank :

“Ces plans sont en quelque sorte semblables aux MOPs, en ce qu'ils contiennent des connaissances générales et normatives des types de situations que l'on rencontre habituellement dans le déroulement d'une activité, et indexent en même temps des cas représentant des variantes du plan...”

[Cañamero, 1995, p. 136]

Nous présenterons ici la structure de plan donnée dans [Cañamero, 1995]. Notons que ces plans contiennent des connaissances utiles pour construire un comportement adéquat (connaissances de planification) et des connaissances utiles à la reconnaissance de plans... Un plan contient donc principalement :

- un ensemble de préconditions, de deux sortes :

- des préconditions d'activation, qui précisent les circonstances dans lesquelles le plan est activable (données utiles pour la reconnaissance),

- des préconditions d'action, qui expriment les conditions d'activation du plan (données utiles pour la planification).

- la liste des tâches typiquement réalisées pour la situation décrite. Ces tâches pointent vers d'autres plans ou vers des actions primitives.

- les modes d'exécution possibles du plan, qui expriment les différentes variantes possibles pour réaliser le plan.

- un ensemble de conditions qui permettent de savoir lorsque le plan est terminé,

- une valeur de croyance mise à jour au fur et à mesure du processus de reconnaissance.

Les plans sont donc organisés en une double hiérarchie :

- une hiérarchie de plans construite à partir des préconditions (spécialisation du contexte dans lequel le plan est activable),

- une hiérarchie de décomposition des plans en sous-plans.

Avant de comparer ce type de représentation à nos graphes de nodules, qui sont aussi une hiérarchie de situations, nous allons présenter une autre représentation, très similaire d'un point de vue structurel : les Blocs proposés par [Boy, 1990].

4.6.3 La surveillance de processus

"Quand une situation est reconnue, elle suggère généralement comment résoudre un problème associé"

[Boy, 1993, p. 38]

La notion de Bloc a été définie par Guy Boy pour représenter des connaissances procédurales, que [Boy, 1988] définit comme "situationnelles". L'approche de G. Boy, qui a été utilisée pour l'assistance à des opérateurs et pour l'aide à la documentation, est donc assez similaire à la nôtre :

"... la représentation par bloc est très adaptée à la construction incrémentale de connaissances opérationnelles"

[Boy, 1990, p. 2.2]

La présentation que nous faisons de la notion de bloc est tirée de [Mathé et Kedar, 1992] qui proposent une extension et une simplification du modèle initial de G. Boy, notamment en supprimant la notion de contexte.

Un bloc est un objet qui est activé si ses préconditions sont remplies. Un bloc contient des actions (l'activation d'autres blocs peut être une action) et deux autres champs : un champ “conditions anormales” et un champ “buts”. Ces deux champs définissent ce qu'il convient de faire en fonction des résultats de l'exécution des actions. La détection d'une condition anormale ou l'accomplissement d'un but peut activer d'autres blocs.

Il y a donc une grande similarité avec les graphes de nodules :

- les champs “conditions anormales” et “buts” sont équivalents à notre champ “règles de choix” : la représentation par bloc a été conçue pour le contrôle de systèmes dynamiques et la distinction entre résultats d'actions attendus et résultats anormaux est très importante dans ce contexte. Toutefois, il s'agit toujours de décider de la marche à suivre une fois que les actions associées à la situation ont été exécutées.

- l'appel de sous-blocs depuis le champ “actions” d'un bloc équivaut au déclenchement d'un sous-graphe.

On peut aussi noter des ressemblances entre les blocs et les plans introduits dans la section précédente :

- les deux objets contiennent un champ “préconditions”,

- ils contiennent un champ qui décompose la tâche à effectuer en faisant appel éventuellement à d'autres objets (le champ “actions” des blocs et la liste des tâches dans les plans).

La grande différence entre les blocs et les plans, d'un côté, et les nodules de situation, de l'autre, est dans la présence d'un champ “préconditions”.

La représentation par nodules oblige en effet à exprimer explicitement le contenu de ces champs “préconditions” dans les règles des nodules précédents. Les connaissances dites “de contrôle” sont donc beaucoup plus simplement exprimées puisqu'elles apparaissent de manière explicite dans les seules “règles de choix”. Le contrôle dans la représentation par blocs ou par plans est plus complexe puisqu'une action donnée peut chercher à activer un bloc qui n'est actuellement pas activable : comment doit alors se comporter le SBC ? Surtout comment préciser ce qu'il convient alors de faire ? Cette “situation précise”, c'est-à-dire le modèle du cas correspondant, peut-il alors vraiment être représenté explicitement ?

Les connaissances du champ “préconditions” sont à notre avis redondantes (et peut-être parfois contradictoires) avec celles contenues dans les autres champs. De plus, la complexité du contrôle diminue la compréhensibilité de la BC et sa révisabilité. Nous défendons l'idée que l'utilisation d'objets uniformes, exploités par une structure de contrôle simple et unique, rend plus facile la compréhension et la

correction de la BC. Prenons pour défendre notre argument l'exemple de l'architecture multi-agents DESIRE ([Kowalczyk et Treur, 1990]) dans laquelle un agent est constitué de la somme de modules indépendants activés et désactivés par un module "superviseur", qui comprend donc les connaissances de contrôle de l'agent. La compréhension de l'un de ces modules nécessite de connaître les connaissances de contrôle qui spécifient à quel moment le module en question est activé, avant quel autre module, etc. Dans notre modèle, il n'y a pas de connaissances de contrôle exprimées séparément des autres connaissances : l'utilisation d'une structure de contrôle unique (l'algorithme de fonctionnement du moteur donné dans la figure 1.6) rend davantage explicite les objets de la BC et le comportement du système.

4.7 Les représentations des tâches

Avant de clore ce chapitre, revenons au domaine de l'acquisition des connaissances. Dans ce domaine, des formalismes simples existent pour représenter les tâches et leur décomposition. On peut citer ainsi la structure de tâches de [Chandrasekaran et Johnson, 1993], mais aussi les langages graphiques qu'on trouve par exemple dans l'environnement MACAO ([Matta et Aussenac-Gilles, 1996]) ou le langage LISA ([Delouis, 1993]). La décomposition d'une tâche en sous-tâches peut être un arbre de décomposition simple, lorsqu'il n'est possible d'associer à chaque tâche qu'une seule méthode...

"Mais dans le cas contraire où il serait possible d'associer à une tâche plusieurs méthodes, on obtiendrait un arbre de décomposition conditionnel et/ou. La difficulté est alors de définir comment on exprime le choix entre les méthodes, et donc comment on exprime et référence le contexte de résolution"

[Causse, 1994, p. 117]

Notons que le langage LISA propose une réponse à ce problème du choix dynamique des méthodes, en associant des connaissances pour effectuer ce choix à chaque objet "but" (tâche) et "méthode". Dans l'environnement MACAO, la structure de "schéma" permet également d'expliciter ces décompositions.

Notre idée est que la représentation par nodules de situation est tout à fait adaptée à l'expression de la décomposition en sous-tâches et à la spécification du choix des méthodes.

En plus du problème du choix des méthodes adaptées, deux autres difficultés apparaissent lorsqu'on veut spécifier comment doit se dérouler une tâche :

- comment spécifier l'ordre des sous-tâches à accomplir ? Une décomposition en sous-tâche peut ne proposer que la liste des sous-tâches à effectuer, sans exprimer de contraintes (exemple : [Chandrasekaran et Johnson, 1993]).

- plus généralement, comment exprimer précisément les connaissances de contrôle ?

En général, ces différents types de connaissance sont exprimés dans des structures différentes. Ainsi, le modèle KADS distingue arbre de tâche et structure de tâches.

L'intérêt d'un formalisme générique comme celui des graphes de nodules est de permettre d'exprimer d'une manière opérationnelle : à la fois la décomposition en sous-tâches, la structure de tâches et la structure d'inférence.

Deux réserves doivent être faites toutefois :

- un graphe de nodules ne représente pas graphiquement les entrées et sorties du modèle ou le flux des données, comme par exemple dans la représentation d'une structure d'inférence KADS. Mais rien n'empêche d'annoter les graphes de manière à faire apparaître ce type d'information.

- il convient auparavant d'introduire dans notre modèle la possibilité d'effectuer des itérations dans les graphes de nodules (extension du modèle proposé dans la section 2.5.3).

4.8 Conclusion

Nous avons construit notre représentation par nodules de situation en suivant la démarche qui a été exposée dans le chapitre 1.

Dans ce chapitre 4, nous avons confronté notre modèle à des formalismes qui présentaient tous des points communs avec nos nodules de situation, tout en gardant chacun leur singularité.

Il ressort de cet état des lieux que notre modèle est le seul qui additionne les caractéristiques suivantes[7] :

- intégration dans la représentation du raisonnement, de la notion d'action comme moyen d'aller chercher les informations utiles dans l'environnement,

- modularité et réutilisation grâce à l'appel possible de sous-graphes,

[7] Notons également qu'aucun des modèles présentés ne satisfaisait tous les objectifs fixés dans la section 1.1…

- représentation explicite du fil de raisonnement, dont chaque étape peut être commentée par le concepteur,

- compréhensibilité des objets de la BC, facilitée par :

  - l'unicité de la structure de contrôle exploitant les objets,

  - la petite taille et la simplicité de ces objets (deux champs),

  - la puissance d'expression qui permet de choisir le niveau d'abstraction des objets (éclatement ou regroupement des nodules).

Conclusion

*"Il apparaît ainsi que l'intelligence humaine ne réside pas tant dans le calcul mais dans la capacité à construire des représentations adéquates des **situations**, ainsi qu'à les **modifier**"*

[Weil-Barais, 1993, p. 50]

Apprendre des situations

La citation ci-dessus concerne l'intelligence humaine et dépasse le cadre de ce travail si l'on comprend l'acte de "construire des représentations adéquates des situations" comme un processus créatif. Mais si l'on interprète cette construction d'abord comme un processus de classification parmi un ensemble préétabli de situations prototypiques, cette phrase illustre exactement le travail présenté dans ce mémoire. Les deux termes que nous avons fait ressortir en gras dans la phrase citée correspondent aux deux axes de notre travail :

- le concept de **situation** nous a servi de fondation pour proposer un formalisme de représentation des connaissances opératoires.

- la nécessité d'être capable de **modifier** de manière adéquate les représentations des situations était une motivation centrale dans notre approche.

Notre travail s'est donc concentré sur le problème suivant, à notre avis fondamental : comment doter un système de la capacité d'apprendre correctement des "situations" incrémentalement et de manière coopérative avec un tuteur /

concepteur, qui conserve un droit de décision sur le contenu de la BC. Dans ce contexte, la BC doit contenir des connaissances opératoires prescriptives, qui décrivent précisément et explicitement ce qu'il convient de faire dans chaque situation.

La notion de situation

La notion de situation est centrale à notre travail dans la mesure où :

- nous proposons un formalisme de représentation des connaissances opératoires, les nodules de situation, qui s'appuie sur la notion de situation.

- nous avons développé des techniques de révision coopérative pour des Bases de Connaissance exprimées dans ce même formalisme.

L'idée essentielle que nous avons voulu défendre peut être résumée de la manière suivante : il est possible d'axer la construction incrémentale de SBC sur la notion de situation, vue comme un **cas en cours de résolution**. Une situation peut être un cas résolu (les nodules terminaux correspondent à des cas résolus), mais c'est une notion plus générale, définie par l'agrégation :

- d'une tâche à effectuer (un projet),
- d'un "moment" dans l'exécution de cette tâche (une étape),
- d'un "état du monde" connu (un environnement).

Contrairement aux notions classiques en IA de cas (résolu) et de règle, la notion de situation permet de décomposer l'exécution d'une tâche. Notre modèle permet de spécifier des contraintes chronologiques sur la manière de résoudre le problème, comme par exemple l'ordre des actions à effectuer. Cette prise en compte du temps rapproche les graphes de nodules de la notion de plan. Notre modèle rejoint aussi les approches en acquisition des connaissances (KADS, Tâches Génériques, etc.) qui mettent l'accent sur la représentation du déroulement d'une tâche. Ainsi, le modèle des nodules de situation "étendu", c'est-à-dire capable de représenter des itérations (voir section 2.5.3), pourrait servir à exprimer les connaissances au niveau "tâche" dans le modèle KADS.

L'objet "nodule de situation" que nous avons introduit pour représenter des situations au sein de la BC, est un grain de connaissance qui possède le savoir-faire attaché à une situation, c'est-à-dire une action sur l'environnement et les moyens "d'inférer la nouvelle situation". Une idée importante attachée à cette représentation est l'idée que les grains de connaissances ne sont jamais indépendants mais toujours fortement liés entre eux. Les grains de connaissances prennent leur signification par ces liens. Les nodules de situation sont finalement une proposition pour représenter les connaissances opératoires de manière modulaire, dans des structures de plus

petite taille que les frames de Marvin Minsky ([Minsky, 1981]) ou les MOPs de Roger Schank ([Schank, 1982a]).

Modélisation incrémentale opérationnelle et révision située

La représentation des connaissances par nodules de situation a pour but d'aider le concepteur de la BC à modéliser les raisonnements / savoir-faire de bout en bout et à les décomposer explicitement. Elle prend place dans une approche d'acquisition incrémentale des connaissances, dont le but est de recenser progressivement et d'intégrer dans la BC l'ensemble des différentes situations significatives d'un savoir-faire. Nous nous sommes fixé un certain nombre de contraintes pour remplir cet objectif, dont :

- la **compréhensibilité** de la BC, objectif qui distingue par exemple notre approche de l'approche Règles Dé-Roulées,

- le fait que la BC soit constamment **opérationnelle**, objectif qui éloigne notre approche de l'approche KADS.

Nous avons finalement proposé le terme “révision située” pour qualifier notre démarche, dans laquelle une préoccupation essentielle est d'être prêt, à tout moment, à réviser la BC. Les caractéristiques de cette démarche peuvent être reformulées de la manière suivante :

- s'appuyer sur un concepteur et sur des cas concrets, détectés notamment par l'application de la BC courante sur des problèmes réels, pour enrichir la BC dès que nécessaire,

- fournir au concepteur un outil de révision coopérative pour l'aider à corriger facilement la BC et de manière complètement maîtrisée.

- entretenir la compréhensibilité de la BC en s'appuyant sur un formalisme adapté et en ayant le souci de réutiliser et de généraliser les savoir-faire déjà représentés.

La représentation des connaissances par nodules de situation s'est avérée fort intéressante pour la révision parce qu'elle fournit un formalisme :

- qui sert de support à la coopération utilisateur / système et permet de réduire la complexité du problème de révision, parce que l'utilisateur est capable de spécifier très précisément des objectifs de correction.

- qui permet par sa structure de faire des corrections locales très facilement (par exemple : une opération de modification d'une règle de choix a des répercussions limitées sur l'ensemble des exemples couverts par le graphe).

De plus, la notion d'action attachée à une situation et dont le comportement doit toujours être explicitement déclaré, permet de calculer pour chaque graphe de

nodules la “couverture du graphe”. Cette notion de couverture est très utile pour la communication entre l'utilisateur et le système de révision puisqu'elle fournit un langage de description d'exemples abstraits, qui permet à l'utilisateur et au système :

- de choisir ensemble le bon niveau de généralisation de la correction à effectuer,
- de valider l'ensemble des opérations de correction effectuées,
- de chercher à poursuivre la généralisation de la correction effectuée.

Ainsi, le processus général de modélisation incrémentale opérationnelle est facilité par l'emploi d'un outil comme REVINOS qui permet au concepteur de se concentrer sur les objets de la BC qu'il est en charge de maintenir (nodules de situation, actions, faits).

A l'intérieur du modèle proposé dans ce travail, deux directions de recherche intéressantes s'ouvrent :

- Le développement d'opérateurs de restructuration et leur prise en compte éventuelle au cours de l'étape de révision. Les opérateurs de restructuration, qui à l'origine ne changent pas la couverture du graphe, peuvent être légèrement modifiés de manière à créer de nouveaux opérateurs de révision utiles.
- Le développement d'un outil “d'analogie dérivationnelle” sera utile pour détecter et proposer de réutiliser des branches de graphes similaires.

Bibliographie

[Aamodt et Plaza, 1994]
Aamodt A., Plaza E., "Case-Based Reasoning: Foundational Issues", Methodological Variations, and System Approaches, *AI Communications* **7**, 1, p. 39-59, mars 1994.

[Agre et Chapman, 1987]
Agre P., Chapman D., "Pengi: An Implementation of A Theory of Activity", in *Proceedings of the 6th National Conference on Artificial Intelligence (AAAI-87)*, p. 268-272, 1987.

[Aikins, 1983]
Aikins J.S., "Prototypical Knowledge for Expert Systems", *Artificial Intelligence* **20,** 2, p. 163-210, 1983.

[Akman et Surav, 1997]
Akman V., Surav M., "The Use of Situation Theory in Context Modeling", *Computational Intelligence* **13**, 3, p. 427-438, 1997.

[Althoff, Maurer et Rehbold, 1990]
Althoff K-D, Maurer F., Rehbold R., "Multiple Knowledge Acquisition Strategies in MOLTKE", in *Current Trends in Knowledge Acquisition, Proceedings of the Fourth European Knowledge Acquisition Workshop (EKAW'90),* p. 21-40, Wielenga B., Boose J., Gaines B., Schreiber G., van Someren M. (Eds.), IOS Press, 1990.

[Angluin, 1988]
Angluin D., "Queries and Concept Learning", *Machine Learning* **2**, p. 319-342, 1988.

[Bareiss, Porter et Wier, 1990]
Bareiss E.R., Porter B.W., Wier C.C, "PROTOS: An Exemplar-Based Learning Apprentice", in *Machine Learning, An Artificial Intelligence Approach, Volume III,* p. 112-127, Kodratoff Y., Michalski R. (Eds.), Morgan Kaufmann, 1990.

[Barwise et Perry, 1983]
Barwise J., Perry J., *Situations and attitudes*, The MIT Press, Cambridge, Massachussets, 1983.

[Bennett, 1985]
Bennett J.S., "ROGET: A Knowledge-Based System for Acquiring the Conceptual Structure of a Diagnostic Expert System", *Journal of Automated Reasoning* **1**, 1, p. 49-84, 1985.

[Bergadano et Giordana, 1990]
Bergadano F., Giordana A., "Guiding Induction with Domain Theories", in *Machine Learning, An Artificial Intelligence Approach, Volume III,* p. 474-492, Kodratoff Y., Michalski R. (Eds.), Morgan Kaufmann, 1990.

[Bergadano et Gunetti, 1993]
Bergadano F., Gunetti D., "An Interactive System to Learn Functional Logic Programs", in *Proceedings of the 13th International Joint Conference on Artificial Intelligence (IJCAI'93)* p. 1044-1049, Chambéry, août 1993.

[Bisson, 1993]
Bisson G., *KBG : Induction de bases de connaissances en logique des prédicats*, Thèse de l'Université Paris-Sud, avril 1993.

[Bobrow et Fraser, 1969]
Bobrow D.G., Fraser B., "An Augmented State Transition Network Analysis Procedure", in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI'69)*, p. 557-567, Walker D., Norton L. (Eds.), Washington, D.C., mai 1969.

[Boose, 1985]
Boose J., "A Knowledge Acquisition Program For Expert Systems Based On Personal Construct Psychology", *International Journal Of Man-Machines Studies* **23**, p. 495-525, 1985.

[Boose, Bradshaw, Koszarek et Shema, 1993]
Boose J., Bradshaw J., Koszarek J., Shema D., "Knowledge Acquisition Techniques For Group Decision Support", *Knowledge Acquisition* **5**, 4, p. 405-447, 1993.

[Boy, 1988]
Boy G., *Assistance à l'opérateur : une approche de l'intelligence artificielle*, Teknea, 1988.

[Boy, 1990]
Boy G., "Acquiring and Refining Indices According To Context", in *Proceedings of the 6th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop (KAW'90)*, p. 2.1-2.14, novembre 1990.

[Boy, 1993]
Boy G., "Knowledge Acquisition In Dynamic Systems: How Can Logicism And Situatedness Go Together?", in *Proceedings of the 7th European Knowledge Acquisition For Knowledge-Based Systems Workshop (EKAW'93)*, p. 23-44, Aussenac N., Boy G., Linster M., Ganascia J.-G., Kodratoff Y., (Eds.), Lecture Notes in Artificial Intelligence 723, Springer Verlag, septembre 1993.

[Brachman, 1979]
Brachman R.J., "On the Epistemological Status of Semantic Networks", in *Associative Networks: Representation and Use of Knowledge by Computers*, p. 3-50, Findler N.V. (Ed.), Academic Press, 1979. Aussi dans *Readings in Knowledge Representation*, Brachman R.J., Levesque H.J (Eds.), Morgan Kaufmann, 1985.

[Breuker, 1994]
Breuker J., "Components of Problem Solving and Types of Problems", in *A future for Knowledge Acquisition, Proceedings of the 8th European Knowledge Acquisition Workshop (EKAW'94)* , p. 118-136, Steels L., Schreiber G., Van de Velde W. (Eds.), Lecture Notes in Artificial Intelligence 867, Springer-Verlag, 1994.

[Breslow et Aha, 1997]
Breslow L., Aha D., "Simplifying Decision Trees: A Survey", *Knowledge Engineering Review* **12**,1, p. 1-40, 1997.

[Brézillon, 1990]
Brézillon P., "Interpretation and rule packet in expert systems", Application to the SEPT expert system, in *Knowledge Based Computer Systems*, p. 78-87, Ramani S., Chandrasekar R., Anjaneyulu K.S.R. (Eds.), Lectures Notes in Artificial Intelligence 444, Springer-Verlag, 1990.

[Brézillon, 1996]
Brézillon P., *Context in Human-Machine Problem Solving: A Survey,* Rapport de Recherche 96/29 du LAFORIA, octobre 1996.

[Brézillon et Abu-Hakima, 1995]
Brézillon P., Abu-Hakima S., "Using Knowledge in its Context: Report on the IJCAI-93 Workshop", Rapport de Recherche 95/18 du LAFORIA, mai 1995.

[Brézillon et Cavalcanti, 1997]
Brézillon P., Cavalcanti M., "Modeling and Using Context: Report on the First International and Interdiciplinary Conference CONTEXT-97", *Knowledge Engineering Review* **12**, 4, p. 1-10, 1997.

[Brooks, 1991]
Brooks R.A., "Intelligence without representation", *Artificial Intelligence* **47**, p. 139-159, 1991.

[Buchanan et Shorliffe, 1984]
Buchanan B., Shortliffe E. (Eds.), *Rule-Based Expert Systems*, Addison-Wesley, 1984.

[Cañamero, 1994]
Cañamero D., "Modelling Plan Recognition for Decision Support", in *A future for Knowledge Acquisition, Proceedings of the 8th European Knowledge Acquisition Workshop (EKAW'94)* , p. 158-177, Steels L., Schreiber G., Van de Velde W. (Eds.), Lecture Notes in Artificial Intelligence 867, Springer-Verlag, 1994.

[Cañamero, 1995]
Cañamero D., *REPLACE: un modèle pour la reconnaissance de plans*, Thèse de l'Université Paris-Sud, juin 1995.

[Cañamero, Delannoy et Kodratoff, 1993]
Cañamero D., Delannoy J.F., Kodratoff Y., "Explications dans un système de reconnaissance de plans pour l'aide à la décision", *Actes des deuxièmes journées Explication du PRC-GDR-IA*, Sophia-Antipolis, juin 1992.

[Carbonara et Sleeman, 1996]
Carbonara L., Sleeman D., "Improving the efficiency of Knowledge Base Refinement", in *Proceedings of the 13th International Conference on Machine Learning (ICML'96)*, p. 78-86, Saitta L. (Ed.), Bari, Italie, juillet 1996.

[Carbonell, 1986]
Carbonell J.G., "Derivational Analogy: A Theory of Reconstruction Problem Solving and Expertise Acquisition", in *Machine Learning: An Artificial Approach*, *Volume II,* p. 371-392, Michalski R.S., Carbonell J.G., Mitchell T. M. (Eds.), Morgan Kaufmann, 1986.

[Carbonell et Gil, 1990]
Carbonell J.G., Gil Y., "Learning by Experimentation: The Operator Refinement Method", in *Machine Learning, An Artificial Intelligence Approach, Volume III,* p. 191-213, Kodratoff Y., Michalski R. (Eds.), Morgan Kaufmann, 1990.

[Carbonell, Knoblock et Minton, 1991]
Carbonell J.G., Knoblock C.A., Minton S., "PRODIGY: An integrated Architecture for Planning and Learning", In *Architectures for Intelligence*, p. 241-278, VanLehn K. (Ed.), Lawrence Erlbaum, 1991.

[Causse, 1994]
Causse K., *MCC: vers l'acquisition de connaissances de contrôle*, Thèse de l'Université Paris-Sud, septembre 1994.

[Cendrowska, 1987]
Cendrowska J., "PRISM: An Algorithm for Inducing Modular Rules", *International Journal Of Man-Machines Studies* **27**, 4, p. 349-370, 1987.

[Chandrasekaran, 1986]
Chandrasekaran B., "Generic tasks in knowledge-based reasoning: High-level building blocks for expert system design", *IEEE Expert* **1**, 3, p. 23-29, 1986.

[Chandrasekaran et Johnson, 1993]
Chandrasekaran B., Johnson T. R., "Generic Tasks and Tasks Structures : History, Critique and New Directions", In *Second Generation Expert Systems*, p. 237-272, David J-M, Krivine J-P, Simmons R. (Eds.), Springer-Verlag, 1993.

[Cheng, Fayyad, Irani et Qian, 1988]
Cheng J., Fayyad U.M., Irani K.B., Qian Z., "Improved Decision Trees: A Generalized Version of ID3", in *Proceedings of the 5th International Conference on Machine Learning (ICML'88)*, p. 100-106, Laird J. (Ed.), Morgan Kaufmann, juin 1988.

[Clancey,1983]
Clancey W.J., "The Epistemology of Rule-Based Expert System", *Artificial Intelligence* **20**, p. 215-251, 1983.

[Clancey, 1985]
Clancey W.J., "Heuristic Classification", *Artificial Intelligence* **27**, p. 289-350, 1985.

[Clancey, 1991a]
Clancey W.J., "The Frame of Reference Problem in the Design of Intelligent Machines", In *Architectures for Intelligence*, p. 357-423, VanLehn K. (Ed.), Lawrence Erlbaum, 1991.

[Clancey, 1991b]
Clancey W.J., "Situated Cognition: Stepping out of Representational Flatland", *AI Communications* **4**, 2/3, p. 109-112, 1991.

[Clancey, 1992]
Clancey W.J., "Model construction operators", *Artificial Intelligence* **53**, p. 1-115, 1992.

[Clancey, 1993a]
Clancey W.J., "Situated Action: A Neuropsychological Interpretation (Response to Vera and Simon)", *Cognitive Science* **17**, p. 87-116, 1993.

[Clancey, 1993b]
Clancey W.J., "The Knowledge Level Reinterpreted: Modeling Socio-Technical Systems", *International Journal of Intelligent Systems* **8**, 1, p. 33-50, 1993.

[Clancey, 1997a]
Clancey W.J., "The Conceptual Nature of Knowledge, Situations and Activities", in *Human and Machine Expertise in Context*, p. 247-291, Feltovich P., Hoffman R., Ford K. (Eds.), 1997.

[Clancey, 1997b]
Clancey W.J., *Situated Cognition, On Human Knowledge and Computer Representations*, Cambridge University Press, 1997.

[Cohen, 1988a]
Cohen H., "How to draw three people in a botanical garden", in *Proceedings of the 7th National Conference on Artificial Intelligence (AAAI'88)*, p. 846-855, août 1988.

[Compton, 1992]
Compton P., "Insight and Knowledge", in *Working Notes of "Cognitive Aspects of Knowledge Acquisition",* p. 57-63, AAAI Spring Symposium Series, Stanford University, mars 1992.

[Compton et Jansen, 1990]
Compton P., Jansen R., "A Philosophical Basis For Knowledge Acquisition", *Knowledge Acquisition* **2**, 3, p. 241-258, 1990.

[Compton, Kang, Preston et Mulholland, 1993]
Compton P., Kang B., Preston P., Mulholland M., "Knowledge Acquisition Without Analysis", in *Proceedings of the 7th European Knowledge Acquisition For Knowledge-Based Systems Workshop (EKAW'93)*, p. 277-299, Aussenac N., Boy G., Linster M., Ganascia J.-G., Kodratoff Y., (Eds.), Lecture Notes in Artificial Intelligence 723, Springer Verlag, septembre 1993.

[Compton, Preston, Kang et Yip, 1994]
Compton P., Preston P., Kang B., Yip T., "Local Patching Produces Compact Knowledge Bases", in *A future for Knowledge Acquisition, Proceedings of the 8th European Knowledge Acquisition Workshop (EKAW'94)* , p. 104-117, Steels L., Schreiber G., Van de Velde W. (Eds.), Lecture Notes in Artificial Intelligence 867, Springer-Verlag, 1994.

[Compton, Preston, Edwards et Kang, 1996]
Compton P., Preston P., Edwards G., Kang B., "Knowledge based systems that have some idea of their limits", in *Proceedings of Banff Knowledge Acquisition For Knowledge-Based Systems Workshop (KAW'96)*, 1996.

[Compton, Edwards, Kang, Lazarus, Malor, Menzies, Preston, Srinivasan et Sammut, 1991]
Compton P., Edwards G., Kang B., Lazarus L., Malor R., Menzies T. Preston P. Srinivasan A., Sammut S., "RDR: Possibilities and Limitations", in *Proceedings of the 7th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop (KAW'91)*, p. 6.1-6.18, octobre 1991.

[Cornuéjols, 1989]
Cornuéjols A., *De l'Apprentissage Incrémental par Adaptation Dynamique : le système INFLUENCE*, Thèse de l'Université Paris-Sud, janvier 1989.

[Cornuéjols, 1996]
Cornuéjols A., "Analogie, principe d'économie et complexité algorithmique", in *Journées Acquisiton, Apprentissage (JAVA'96)*, p. 233-247, Sète, 1996.

[Craw et Sleeman, 1990]
Craw S., Sleeman D., "Automating the Refinement of Knowledge-Based Systems", in *Proceedings of the 9h European Conference on Artificial Intelligence (ECAI'90)*, p. 167-172, Pitman, Stockholm, 1990.

[Craw et Sleeman, 1995]
Craw S., Sleeman D., *Knowledge-based Refinement of Knowledge Based Systems*, Technical Report AUCS/TR9505, University of Aberdeen, 1995.

[David, Krivine et Ricard, 1993]
David J-M, Krivine J-P, Ricard B., "Building and Maintaining Large Knowledge-Based System from a 'Knowledge Level' Perspective: the DIVA Experiment", In *Second Generation Expert Systems*, p. 376-401, David J-M, Krivine J-P, Simmons R. (Eds.), Springer-Verlag, 1993.

[Davis, 1979]
Davis R., "Interactive Transfer of Expertise: Acquisition of New Inference Rules", *Artificial Intelligence* **12**, p. 121-157, 1979.

[Davis et Buchanan, 1977]
Davis R., Buchanan B., "Meta-Leval Knowledge-Overview and Applications", in *Proceedings of the Fifth International Joint Conference on Artificial Intelligence (IJCAI'77)* , p. 920-927, Cambrige, Massachusetts, août 1977.

[Davis, Buchanan et Shortliffe, 1977]
Davis R., Buchanan B., Shortliffe E., "Production Rules as a Representation for a Knowledge-Based Consultation Program", *Artificial Intelligence* **8**, p. 15-45, 1977.

[DeJong, 1981]
Dejong G., "Generalizations Based on Explanations", in *Proceedings of the Seventh International Joint Conference on Artificial Intelligence (IJCAI'81)*, p. 67-69, Vancouver, août 1981.

[DeJong et Mooney, 1986]
Dejong G., Mooney R., "Explanation-Based Learning : An Alternative View", in *Machine Learning* **1,** 2, p. 145-176, 1986.

[Dent, Boticario, McDermott, Mitchell et Zabowski, 1992]
Dent L., Boticario J., McDermott J., Mitchell T., Zabowski D., "A Personal Learning Apprentice", in *Proceedings of the 10th National Conference on Artificial Intelligence (AAAI'92)*, p. 96-103, juillet 1992.

[De Kleer, 1986]
De Kleer J., "An Assumption-Based TMS", *Artificial Intelligence* **28,** 1, p. 127-162, 1986.

[De Raedt et Bruynooghe, 1992]
De Raedt L., Bruynooghe M., "Interactive Concept-Learning and Constructive Induction by Analogy", *Machine Learning* **8**, 2, p. 107-150, March 1992.

[Devlin, 1991]
Devlin K., *Logic and information*, Cambridge University Press, 1991.

[diSessa, 1987]
diSessa A., "Artificial worlds and real experience", in *Artificial Intelligence and Education Volume 1, Learning Environments and Tutoring Systems*, p. 55-77, Lawler R.W., Yazdani M. (Eds.), Alex Publishing, Norwood, NJ, 1987

[diSessa, 1993]
diSessa A., "Towards an Epistemology of Physics", *Cognition and Instruction* **10**, 2/3, p. 105-225, 1993.

[Donoho et Wilkins, 1994]
Donoho S., Wilkins D., "ODYSSEUS2: Addressing the Challenges of Apprenticeship", in *Proceedings of the 8th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop*, p. 14.1-14.18, January 1994.

[Eshelman, Ehret, MacDermott et Tan, 1987]
Eshelman L., Ehret D., MacDermott J., Tan M., "MOLE: a tenacious knowledge-acquisition tool", *International Journal Of Man-Machines Studies* **26**, p. 41-54, 1987.

[Feigenbaum, 1977]
Feigenbaum E., "The Art of Artificial Intelligence: Themes and case studies of Knowledge Engineering", in *Proceedings of the Fifth International Joint Conference on Artificial Intelligence (IJCAI'77)*, p. 1014-1029, Cambrige, Massachusetts, août 1977.

[Ferber, 1995]
Ferber J., *Les systèmes multi-agents, vers une intelligence collective*, InterEditions, Paris, 1995.

[Fikes et Keller, 1985]
Fikes R., Keller T., "The Role of Frame-Based Representation in Reasoning", *Communications of the ACM* **28**, 9, p. 906-920, septembre 1985.

[Fikes, Hart et Nilsson, 1972]
Fikes R.E., Hart P.E., Nilsson N.J., "Learning and Executing Generalized Robot Plans", *Artificial Intelligence* **3**, 2, p. 251-288, 1972.

[Fisher, 1987]
Fisher D.H., "Knowledge Acquisition Via Incremental Conceptual Clustering", *Machine Learning* **2**, p. 139-172, 1987.

[Floyd , 1984]
Floyd C., "A Systematic Look at Prototyping", in *Approaches to Prototyping,* p. 1-18, Budde R., Kuhlenkamp K., Mathiassen L., Züllighiven H., (Eds.), Springer-Verlag, 1984.

[Fodor, 1983]
Fodor J., *The Modularity Of Mind*, The MIT Press, Cambridge, Massachussets, 1983, trad. fr.: Gerschenfeld A., *La modularité de l'esprit*, Paris, Editions de Minuit, 1986.

[Gadamer, 1976]
Gadamer H-G, *Vérité et méthode*, Paris, Editions du Seuil, collection “L'ordre philosophique”, 1976.

[Gaines, 1991]
Gaines B.R., "Induction and visualization of rules with exceptions", in *Proceedings of the 7th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop (KAW'91)*, p. 7.1-7.17, octobre 1991.

[Gaines, 1996]
Gaines B.R., "Transforming Rules and Trees into Comprehensible Knowledge Structures", in *Advances in Knowledge Discovery and Data Mining*, p. 205-226, Fayyad U.M., Piatetsky-Shapiro G., Smyth P., Uthurusamy R. (Eds.), The MIT Press, Cambridge, Massachussets, 1996.

[Gaines et Compton, 1995]
Gaines B.R., Compton P., "Induction of Ripple-Down Rules Applied to Modeling Large Databases", *Journal for Intelligent Information Systems* **5**, 3, p. 211-228, 1995.

[Gaines et Shaw, 1996]
Gaines B.R., Shaw M.L., "A Networked, Open Architecture Knowledge Management System", in *Proceedings of Banff Knowledge Acquisition For Knowledge-Based Systems Workshop (KAW'96)*, 1996.

[Ganascia, 1990]
Ganascia J-G., *L'âme machine, les enjeux de l'intelligence artificielle*, Editions du Seuil, 1990.

[Gaudel, Marre, Schlienger et Bernot, 1996]
Gaudel M.G., Marre B., Schlienger F., Bernot G., *Précis de Génie Logiciel*, Masson, Paris, 1996.

[Genesereth et Nilsson, 1987]
Genesereth M.R., Nilsson N.J., *Logical Foundations of Artificial Intelligence*, Morgan Kaufmann, 1987.

[Georgeff, 1985]
Georgeff M., "Reasoning About Procedural Knowledge", in *Proceedings of the 4th National Conference on Artificial Intelligence (AAAI'85)*, 1985.

[Georgeff et Bonollo, 1983]
Georgeff M., Bonollo U., "Procedural Expert Systems", in *Proceedings of the Eight International Joint Conference on Artificial Intelligence (IJCAI'83),* p. 151-157, Bundy A. (Ed.), Karlsruhe, août 1983.

[Georgeff, Lansky et Bessiere, 1985]
Georgeff M., Lansky A., Bessiere P., "A Procedural Logic", in *Proceedings of the Nine International Joint Conference on Artificial Intelligence (IJCAI'85),* p. 673-680, Joshi A. (Ed.), Los Angeles, août 1985.

[Ginsberg, 1988a]
Ginsberg A., *Automatic Refinement of Expert System Knowledge Bases*, Pitman, 1988.

[Ginsberg, 1988b]
Ginsberg A., "Knowledge-Base Reduction: A New Approach to Checking Knowledge-Bases for Inconsistency & Redundancy", in *Proceedings of the 7th National Conference on Artificial Intelligence (AAAI'88),* p. 585-589, août 1988.

[Ginsberg, 1988c]
Ginsberg A., "Theory Revision via Prior Operationalization", in *Proceedings of the 7th National Conference on Artificial Intelligence (AAAI'88)*, p. 590-595, août 1988.

[Ginsberg, 1990]
Ginsberg A., "Theory Reduction, Theory Revision and Retranslation", in *Proceedings of the 8th National Conference on Artificial Intelligence (AAAI'90)*, p. 777-782, 1990.

[Grant, 1994]
Grant S., "Modeling complex cognition: contextual modularity and transitions", in *Proceedings of the 4th International Conference on User Modeling*, p. 157-162, Hyannis, USA, août 1994.

[Gruber, 1989a]
Gruber T., Automated Knowledge Acquisition of Strategic Knowledge, *Machine Learning* **4**, p. 293-336, 1989.

[Gruber, 1989b]
Gruber T., *The Acquisition of Strategic Knowledge*, Academic Press, 1989.

[Haton, Bouzid, Charpillet, Haton, Lâasri, Lâasri , Marquis, Mondot et Napoli, 1991]
Haton J-P, Bouzid N., Charpillet F., Haton M-C, Lâasri B., Lâasri H., Marquis P., Mondot T., Napoli A., *Le raisonnement en intelligence artificielle*, InterEditions, 1991.

[Hewitt, Bishop et Steiger, 1973]
Hewitt C., Bishop P., Steiger R., "A Universal Modular ACTOR Formalism for Artificial Intelligence", in *Proceedings of the 3th International Joint Conference on Artificial Intelligence (IJCAI'73),* p. 235-245, Stanford, Californie, août 1973.

[Hofstadter, 1988]
Hofstadter D., "Eveil après le rêve booléen", ou "sous-cognition computationnelle", in *Ma Thémagie*, Paris, InterEditions, 1988.

[Jackson, 1990]
Jackson P., *Introduction to Expert Systems*, Second Edition, Addison-Wesley, 1990.

[Jaspers, 1932]
Jaspers K., *Die Geistige Situation der Zeit*, 1932, Berlin-New-York, Walter de Gruyter, 1979.

[Kaelbling, 1986]
Kaelbling L.P., "An Architecture for Intelligent Reactive Systems", in *Proceedings of the 1986 Workshop "Reasoning About Actions & Plans"*, p. 395-411, Georgeff M., Lansky A. (Eds.), Morgan Kaufmann, 1987, Timberline, Oregon, juillet 1986.

[Karbach, Linster et Voss, 1990]
Karbach W., Linster M., Voss A., "Models, methods, roles and tasks: many labels-one idea ?", *Knowledge Acquisition* **2**, 4, p. 279-299, 1990.

[Kang et Compton, 1992]
Kang B., Compton P., "Towards a Process Memory", in *Working Notes of "Cognitive Aspects of Knowledge Acquisition"*, p. 139-146, AAAI Spring Symposium Series, Stanford University, mars 1992.

[Kang, Compton et Preston, 1998]
Kang B., Compton P., Preston P., "Simulated Expert Evaluation of Multiple Classification Ripple Down Rules", in *Proceedings of the 11th Banff Knowledge Acquisition For Knowledge-Based Systems Workshop (KAW'98)*, p. EVAL-4, 1998.

[Kayser, 1997]
Kayser D., *La représentation des connaissances*, Hermès, 1997.

[Kelly, 1955]
Kelly G.A., *The Psychology of Personal Constructs*, Norton, 1955.

[Klinker, Bhola, Dallemagne, Marques et MacDermott, 1991]
Klinker G., Bhola C., Dallemagne G., Marques D., MacDermott J., "Usable and reusable programming constructs", *Knowledge Acquisition* **3**, 2, p. 117-135, 1991.

[Kodratoff, 1986]
Kodratoff Y., *Leçons d'apprentissage symbolique*, Cépaduès Editions, 1986.

[Kodratoff, 1994]
Kodratoff Y., "Guest Editor's Introduction: The Comprehensibility Manifesto", *AI Communications* **7**, 2, p. 83-85, 1994.

[Kodratoff et Barès, 1991]
Kodratoff Y, Barès M., *Base terminologique de l'intelligence artificielle,* Technique et documentation - Lavoisier, 1991.

[Kodratoff et Ganascia, 1986]
Kodratoff Y, Ganascia J-G, "Improving the Generalization Step in Learning", in *Machine Learning: An Artificial Approach*, *Volume II,* p. 215-244, Michalski R.S., Carbonell J.G., Mitchell T. M. (Eds.), Morgan Kaufmann, 1986.

[Kolodner, 1993]
Kolodner J., *Case-Based Reasoning*, Morgan Kaufmann, 1993.

[Kohavi, 1993]
Kohavi R., "Bottom-Up Induction of Oblivious Read-Once Decision Graphs", in *Proceedings of the European Conference on Machine Learning (ECML'94)*, p. 154-169, Bergadano F., De Raedt L. (Eds.), Lecture Notes in Artificial Intelligence 784, Springer-Verlag, Catania, Italie, avril 1994.

[Kowalczyk et Treur, 1990]
Kowalczyk W., Treur J., "On the use of a formalized generic task model in knowledge acquisition", in *Current Trends in Knowledge Acquisition, Proceedings of the Fourth European Knowledge Acquisition Workshop (EKAW'90),* p. 198-221, Wielenga B., Boose J., Gaines B., Schreiber G., van Someren M. (Eds.), IOS Press, 1990.

[Krivine et David, 1988]
Krivine J.-P., David J.-M., "Acquisition de connaissances expertes à partir de situations-types", in *huitièmes journées internationales "Les systèmes experts et leurs applications"*, Avignon, EC2, 1988.

[Krivine et David, 1991]
Krivine J.-P., David J.-M., "L'acquisition des connaissances vue comme un processus de modélisation : méthodes et outils", *Intellectica* **12**, p. 101-137, 1991.

[Kuhn, 1970]
Kuhn T., *The structure of scientific revolutions*, University of Chigaco Press, Second Edition, 1970, trad. fr.: Meyer L., *La structure des révolutions scientifiques*, Flammarion, 1983.

[Laurière, 1986]
Laurière J-L, *Intelligence Artificielle, résolution de problèmes par l'Homme et la machine,* Eyrolles, 1986.

[Lave, 1988]
Lave J., *Cognition in Practice*, Cambridge University Press, 1988.

[Lebbe et Vignes, 1997]
Lebbe J., Vignes R., "Génération de graphes d'identification à partir de description de concepts", in *Induction symbolique et numérique à partir de données*, p. 193-239, Kofratoff Y., Diday E., (Eds.), Cépaduès Editions, 1997.

[Le Ny, 1989]
Le Ny J-F, *Science cognitive et compréhension du langage*, Paris, Presses Universitaires de France, collection "Le psychologue", 1989.

[Lemaire, Safar et Yonnet, 1996]
Lemaire B., Safar B., Yonnet C., "L'acquisition des connaissances d'explication", in *Acquisition et Ingénierie des Connaissances, tendances actuelles*, p. 367-384, Reynaud C., Laublet P., Aussenac-Gilles N. (Eds.), Cépaduès Editions, 1996.

[Lévy, 1990]
Lévy P., *Les technologies de l'intelligence*, La Découverte, 1990.

[Lieberman, 1996]
Lieberman H., "Knowledge Acquisition By Demonstration", in *Papers from the AAAI Symposium "Acquisition, Learning & Demonstration: Automating Tasks for Users"*, p. 62-65, 1996

[Linster, 1993]
Linster M., "Explicit and operational models as a basis for second generation Knowledge Acquisition tools", In *Second Generation Expert Systems*, p. 465-494, David J-M, Krivine J-P, Simmons R. (Eds.), Springer-Verlag, 1993.

[MacCarthy et Hayes, 1969]
MacCarthy J., Hayes P. J., "Some philosophical problems from the standpoint of Artificial Intelligence", *Machine Intelligence* **4**, p. 463-502, Meltzer B., Michie D. (Eds.), 1969.

[MacClelland, 1991]
MacClelland J.L., "Nature, Nurture, and Connections: Implications of Connectionist Models for Cognitive Development", In *Architectures for Intelligence*, p. 41-73, VanLehn K. (Ed.), Lawrence Erlbaum, 1991.

[MacDermott, 1988]
MacDermott J., "Preliminary Steps Towards a Taxonomy of Problem-Solving Methods", in *Automating Knowledge Acquisition for Expert Systems*, p. 225-256, Marcus S. (Ed.), Kluwer Academic Press, 1988.

[Masini, Napoli, Colnet, Léonard et Tombre, 1989]
Masini G., Napoli A., Colnet D., Léonard D., Tombre K., *Les langages à objets*, InterEditions, 1989.

[Marcus, 1988]
Marcus S., "SALT: A knowledge acquisition tool for propose-and revise systems", in *Automating Knowledge Acquisition for Expert Systems*, p. 81-124, Marcus S. (Ed.), Kluwer Academic Press, 1988.

[Mathé et Kedar, 1992]
Mathé N., Kedar S., "Increasingly Automated Procedure Acquisition in Dynamic Systems", in *Proceedings of Banff Knowledge Acquisition For Knowledge-Based Systems Workshop (KAW'92)*, octobre 1992. Aussi Technical Report FIA-92-23, NASA Ames Research Center, juin 1992.

[Matta et Aussenac-Gilles, 1996]
Matta N., Aussenac-Gilles N., "Le schéma du modèle conceptuel, étape dans la modélisation des connaissances", in *Acquisition et Ingénierie des Connaissances, tendances actuelles*, p. 29-48, Reynaud C., Laublet P., Aussenac-Gilles N. (Eds.), Cépaduès Editions, 1996.

[Maurer, 1992]
Maurer F., "Knowledge Base Maintenance and Consistency Checking in MOLTKE/HyDi", in *Current Developments in Knowledge Acquisition, Proceedings of the 6th European Knowledge Acquisition Workshop (EKAW'92)*, p. 337-352, Wetter T., Althoff K.-D., Boose J., Gaines B., Linster M., Schmalhofer F. (Eds.), Lecture Notes in Artificial Intelligence 599, Springer-Verlag, mai 1992.

[Menzies, 1996]
Menzies T., "Assessing Responses to Situated Cognition", in *Proceedings of Banff Knowledge Acquisition For Knowledge-Based Systems Workshop (KAW'96)*, 1996.

[Michalski, 1983]
Michalski R.S., "A Theory and Methodology Of Inductive Learning", in *Machine Learning: An Artificial Intelligence Approach*, p. 162-172, Michalski R.S., Carbonell J., Mitchell T. (Eds.), Morgan Kaufmann, 1983.

[Miller et Goldstein, 1977]
Miller M., Goldstein I., "Structured Planning and Debugging", in *Proceedings of the Fifth International Joint Conference on Artificial Intelligence (IJCAI'77)*, pp 773-779, Cambrige, Massachusetts, août 1977.

[Minsky, 1981]
Minsky M., "A Framework for Representing Knowledge", in *Mind Design,* p. 95-128, Haugeland J. (Ed.), The MIT Press, Cambridge, Massachussets, 1981.

[Mitchell, 1977]
Mitchell T.M., "Version spaces: A Candidate Elimination Approach to Rule Learning", in *Proceedings of the Fifth International Joint Conference on Artificial Intelligence (IJCAI'77),* p. 305-310, Cambrige, Massachusetts, août 1977.

[Mitchell, 1983a]
Mitchell T.M., "Generalization as Search", *Artificial Intelligence* **18**, 2, p. 203-226, 1983.

[Mitchell, 1983b]
Mitchell T.M., "Learning and Problem Solving", in *Proceedings of the Eight International Joint Conference on Artificial Intelligence (IJCAI'83),* p. 1139-1151, Bundy A. (Ed.), Karlsruhe, août 1983.

[Mitchell, Keller et Kedar-Cabelli, 1986]
Mitchell T.M., Keller R.M., Kedar-Cabelli S., "Explanation-Based Generalization : An Unifying View", *Machine Learning* **1,** p. 47-80, 1986.

[Mitchell, Mahadevan et Steinberg, 1990]
Mitchell T.M., Mabadevan S., Steinberg L., "LEAP: A Learning Apprentice For VLSI Design", in *Machine Learning: An Artificial Approach*, *Volume III,* p. 271-301, Kodratoff Y., Michalski R. (Eds.), Morgan Kaufmann, 1990.

[Mitchell, Utgoff et Banerji, 1983]
Mitchell T.M., Utgoff P.E., Banerji R., "Learning by Experimentation: Acquiring and Refinig Problem-Solving Heuristics", in *Machine Learning: An Artificial Intelligence Approach*, p. 163-190, Michalski R.S., Carbonell J., Mitchell T. (Eds.), Morgan Kaufmann, 1983.

[Morik, 1989]
Morik K., "Sloppy modeling", in *Knowledge Representation and Organization in Machine Learning*, p. 107-134, Morik K. (Ed.), Lecture Notes in Artificial Intelligence 347, Springer-Verlag, 1989.

[Morik, Wrobel, Kietz et Emde, 1993]
Morik K., Wrobel S., Kietz J.U., Emde W., *Knowledge Acquisition and Machine Learning*, Academic Press, London, 1993.

[Mostow, 1990]
Mostow J., "Design by Derivational Analogy: Issues in the Automated Replay of Design Plans", in *Machine Learning: Paradigms and Methods*, p. 119-184, Carbonell J. (Ed.), The MIT Press, Cambridge, Massachussets, 1990.

[Musen, 1989]
Musen M., "Automated Support for Building and Extending Expert Models", *Machine Learning* **4**, p. 347-375, 1989.

[Nédellec, 1991]
Nédellec C., "Smallest Generalization Step Strategy", in *Proceedings of the 8th International Workshop (ML'91)*, p. 529-533, Birnbaum L.A., Collins G.C. (Eds.), 1991.

[Nédellec, 1994]
Nédellec C., *APT, apprentissage interactif de règles de résolution de problèmes en présence de théorie du domaine*, Thèse de l'Université Paris-Sud, mars 1994.

[Nédellec, 1996]
Nédellec C., "APT, un système d'apprentissage coopératif", in *Acquisition et Ingénierie des Connaissances, tendances actuelles*, p. 307-327, Reynaud C., Laublet P., Aussenac-Gilles N. (Eds.), Cépaduès Editions, 1996.

[Nédellec, 1998]
Nédellec C., *Cooperative revision with APT system*, Deliverable projet ESPRIT Inductive Logic Programming 2.

[Nédellec et Causse, 1992]
Nédellec C., Causse K., "Knowledge Refinement using Knowledge Acquisition and Machine Learning Methods", in *Current Developments in Knowledge Acquisition, Proceedings of the 6th European Knowledge Acquisition Workshop (EKAW'92)*, p. 171-190, Wetter T., Althoff K.-D., Boose J., Gaines B., Linster M., Schmalhofer F. (Eds.), Lecture Notes in Artificial Intelligence 599, Springer-Verlag, mai 1992.

[Nédellec, Rouveirol, Adé, Bergadano et Tausend, 1996]
Nédellec C., Rouveirol C., Adé H., Bergadano F., Tausend B., "Declarative Bias in Inductive Logic Programming", in *Advances in Inductive Logic Programming*, p. 82-103, De Raedt L. (Ed.), IOS Press, 1996.

[Newell, 1982]
Newell A., "The Knowledge Level", *Artificial Intelligence* **18,** p. 87-127, 1982.

[Newell et Simon, 1972]
Newell A., Simon H., *Human Problem Solving*, Prentice-Hall, Englewood Cliffs, NJ, 1972.

[Ourston et Mooney, 1990]
Ourston D., Mooney R.J., "Changing the Rules : A Comprehensive Approach To Theory Refinement", in *Proceedings of the 8th National Conference on Artificial Intelligence (AAAI'90)*, p. 815-820, août 1990.

[Ourston et Mooney, 1994]
Ourston D., Mooney R., "Theory refinement combining analytical and empirical methods", *Artificial Intelligence* **66**, p. 311-344, 1994.

[Panasiuk, 1998]
Panasiuk A., Communication personnelle.

[Patil, Szolovits et Schwartz, 1981]
Patil R.S., Szolovits P., Schwartz W.B., "Causal Understanding of Patient Illness in Medical Diagnosis", in *Proceedings of the Seventh International Joint Conference on Artificial Intelligence (IJCAI'81)* , p. 893-899, Vancouver, août 1981.

[Pierrel, 1987]
Pierrel J.-M., *Dialogue oral homme-machine*, Hermès, Paris, 1987.

[Pitrat, 1993]
Pitrat J., *Penser l'informatique autrement*, Hermès, Paris, 1993.

[Poittevin, 1993]
Poittevin L., *Une proposition de représentation explicite des connaissances : les nodules de situation*, Note de Recherche D.E.A. Psychologie "Modélisation cognitive", L.E.A.C.M., Université Lyon 2.

[Poittevin, 1996]
Poittevin L., "Comprendre pour réviser : comment la situation peut-elle s'expliquer ?", in *Actes des 3èmes journées Explication*, p. 231-243, Sophia-Antipolis, 1996.

[Poittevin, 1997]
Poittevin L., "REVINOS : un outil de révision interactive s'appuyant sur la notion de situation", in *Actes des Journées Ingénierie des Connaissances et Apprentissage Automatique (JICAA'97)*, p. 467-580, Roscoff, 1997.

[Poittevin, 1998]
Poittevin L., "Building Knowledge Bases with Situations to Help the Cooperative Revision", in *Proceedings of the 11th Banff Knowledge Acquisition For Knowledge-Based Systems Workshop (KAW'98)*, p. KAT-3, 1998.

[Popper, 1979]
Popper K., *Objective Knowledge, An Evolutionary Approach*, Revised Edition, Clarendon Press, Oxford, 1979.

[Preston, Compton, Edwards et Kang, 1996]
Preston P., Compton P., Edwards G., Kang B., "An Implementation of Multiple Classification Ripple Down Rules", in *Proceedings of Banff Knowledge Acquisition For Knowledge-Based Systems Workshop (KAW'96)*, 1996.

[Puerta, Egar, Tu et Musen, 1992]
Puerta A.R., Egar J.W., Tu S.W., Musen M.A., "A multiple-method knowledge-acquisition shell for the automatic generation of knowledge-acquision tools", *Knowledge Acquisition* **4**, 2, p. 171-196, 1992.

[Quinlan, 1986]
Quinlan J.R., "Induction of Decision Trees", *Machine Learning* **1**,1, p. 81-106, 1986.

[Quinlan, 1990]
Quinlan, J.R., "Learning Logical Definitions from Relations", *Machine Learning* **5**, p. 239-266, 1990.

[Rademakers et Vanwelkenhuysen, 1993]
Rademakers P., Vanwelkenhuysen J., "Generic Models and Their Support in Modeling Problem Solving Behavior", In *Second Generation Expert Systems*, p. 350-375, David J-M, Krivine J-P, Simmons R. (Eds.), Springer-Verlag, 1993.

[Reinders, Vinkhuyzen, Voss, Akkermans, Balder, Bartsch-Spörl, Bredeweg, Drouven, van Harmelen, Karbach, Karssen, Schreiber et Wielenga, 1991]
Reinders M., Vinkhuyzen E., Voss A., Akkermans H., Balder J., Bartsch-Spörl B., Bredeweg B., Drouven U., van Harmelen F., Karbach W., Karssen Z., Schreiber G., Wielenga B., "A Conceptual Modelling Framework for Knowledge-Level Reflection", *AI Communications* **4**, 2/3, p. 74-87, 1991.

[Reiter, 1980]
Reiter R., "A Logic For Default Reasoning", *Artificial Intelligence* **13**, 1980.

[Richard, 1990]
Richard J-F, *Les activités mentales : comprendre, raisonner, trouver des solutions*, Armand Colin, Paris, 1990.

[Riddle, 1984]
Riddle W.E., "Advancing the State of Art in Software System Prototyping", in *Approaches to Prototyping,* p. 19-26, Budde R., Kuhlenkamp K., Mathiassen L., Züllighiven H. (Eds.), Springer-Verlag, 1984.

[Riesbeck et Schank, 1992]
Riesbeck C., Schank R., *Inside Case-Based Reasoning*, Lawrence Erlbaum Associates, 1989.

[Rivest, 1987]
Rivest R.L., "Learning Decision Lists", in *Machine Learning* **2**, 3, p. 229-246, 1987.

[Robinet, 1990]
Robinet J-F, article "Situation", in *Encyclopédie Philosophique Universelle, Dictionnaire des notions philosophiques,* Tome 2, Presses Universitaires de France, 1990.

[Rosenchein et Kaelbling, 1995]
Rosenschein S., Kaelbling L.P., "A Situated View of Representation and Control", *Artificial Intelligence* **73**, 1/2, p. 149-173, 1995.

[Russell et Norvig, 1995]
Russell S., Norvig P., *Artificial Intelligence, A Modern Approach*, Prenctice-Hall, 1995.

[Sabah, 1990]
Sabah G., *L'intelligence artificielle et le langage*, Vol. 1 et 2, Deuxième édition, Hermès, Paris, 1990.

[Saitta, Botta et Neri, 1993]
Saitta L., Botta M, Neri P., "Multistrategy Learning and Theory Revision", *Machine Learning* **11**, 2/3, p. 153-172, 1993.

[Sammut et Banerji, 1986]
Sammut C., Banerji R.B., "Learning Concepts By Asking Questions", in *Machine Learning: An Artificial Approach*, *Volume II,* p. 167-192, Michalski R.S., Carbonell J.G., Mitchell T. M. (Eds.), Morgan Kaufmann, 1986.

[Sandberg et Wielenga, 1991]
Sandberg J., Wielenga B., "How situated is cognition ?", in *Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI'91)* , p. 341-346, Sydney, Australie, août 1991.

[Schank, 1982a]
Schank R., *Dynamic Memory: A Theory of Reminding and Learning in Computers and People*, Cambridge University Press, 1982.

[Schank, 1982b]
Schank R., "Looking at Learning", in *Proceedings of the European Conference on Artificial Intelligence (ECAI'82),* p. 11-18, Orsay, juillet 1982.

[Schlimmer et Fischer, 1986]
Schlimmer J., Fischer D., "A Case Study of Incremental Concept Induction", in *Proceedings of the 5th National Conference on Artificial Intelligence (AAAI'86)*, p. 496-501, août 1986.

[Schoppers, 1987]
Schoppers M.J., "Universal plans for Reactive Robots in Unpredictable Environments", in *Proceedings of the 1Oth International Joint Conference on Artificial Intelligence (IJCAI'87)*, p. 1039-1046, McDermott J. (Ed.), Milan, août 1987.

[Schoppers, 1995]
Schoppers M.J., "The Use Of Dynamics in an Intelligent Controller for a Space Faring Rescue Robot", *Artificial Intelligence* **73**, 1/2, p. 175-230, 1995.

[Schreiber, Wielenga et Breuker, 1993]
Schreiber A.T., Wielenga B.J., Breuker J.A. (Eds.), *KADS: a principled approach to Knowledge-Based System Development*, Academic Press, 1993.

[Shadbolt et Wielenga, 1990]
Shadbolt N., Wielenga B., "Knowledge-Based Knowledge Acquisition : The next generation of support tools", in *Current Trends in Knowledge Acquisition, Proceedings of the Fourth European Knowledge Acquisition Workshop (EKAW'90),* p. 318-338, Wielenga B., Boose J., Gaines B., Schreiber G., van Someren M. (Eds.), IOS Press, 1990.

[Shapiro, 1981]
Shapiro E.Y., "An Algorithm that Infers Theories from Facts", in *Proceedings of the Seventh International Joint Conference on Artificial Intelligence (IJCAI'81)*, p. 446-451, Vancouver, août 1981.

[Shaw et Gaines, 1993]
Shaw M.L., Gaines B.R., "Personal Construct Psychology Foundations for Knowledge Acquisition", in *Proceedings of the 7th European Knowledge Acquisition For Knowledge-Based Systems Workshop (EKAW'93)*, p. 256-276, Aussenac N., Boy G., Linster M., Ganascia J.-G., Kodratoff Y., (Eds.), Lecture Notes in Artificial Intelligence 723, Springer Verlag, septembre 1993.

[Shaw et Gaines, 1996]
Shaw M.L., Gaines B.R., "WebGrid: Knowledge Modeling and Inference through the World Wide Web", in *Proceedings of Banff Knowledge Acquisition For Knowledge-Based Systems Workshop (KAW'96)*, 1996.

[Smith et Rosenbloom, 1990]
Smith B.D., Rosenbloom P.S., "Incremental Non-Backtracking Focusing: A Polynomially Bounded Generalization Algorithm for Version Space", in *Proceedings of the 8th National Conference on Artificial Intelligence (AAAI'90)*, p. 848-853, août 1990.

[Smith, Winston, Mitchell et Buchanan, 1985]
Smith R., Winston H., Mitchell T., Buchanan B., "Representation and Use of Explicit Justifications for K B Refinement", in *Proceedings of the Nine International Joint Conference on Artificial Intelligence (IJCAI'85),* p. 673-680, Joshi A. (Ed.), Los Angeles, août 1985.

[Smolenski, 1992]
Smolenski P., "IA connexionniste, IA symbolique et cerveau", in *Introduction aux sciences cognitives,* p. 77-106, Gallimard, 1992.

[Sommer, 1995]
Sommer E., "An Approach to Quantifying the Quality of Induced Theories", in *Proceedings of IJCAI-95 Workshop "Machine Learning and Comprehensibility"*, p. 93-107, Kodratoff Y., Nédellec C. (Eds.), Montréal, août 1995.

[Steels, 1990]
Steels L., "Components of Expertise", *AI magazine* **11**, 2, p. 28-49, 1990.

[Steels, 1993]
Steels L., "The Componentiel framework and its role in reusability", In *Second Generation Expert Systems*, p. 273-298, David J-M, Krivine J-P, Simmons R. (Eds.), Springer-Verlag, 1993.

[Tecuci, 1992]
Tecuci G., "Cooperation in Knowledge Base Refinement", in *Proceedings of the 9th International Workshop on Machine Learning (ML'92)*, p. 445-450, Sleeman D., Edwards P. (Eds.), Morgan Kaufmann, 1992.

[Tecuci et Kodratoff, 1990]
Tecuci G., Kodratoff Y., "Apprenticeship Learning in Imperfect Domain Theories", *Machine Learning, An Artificial Intelligence Approach, Volume III,* p. 514-551, Kodratoff Y., Michalski R. (Eds.), Morgan Kaufmann, 1990.

[Terpstra, van Heijst, Shadbolt et Wielenga, 1993]
Terpstra P., van Heijst G., Shadbolt N., Wielenga B., "Knowledge Acquisition Process Support Through Generalised Directive Models", In *Second Generation Expert Systems*, p. 428-455, David J-M, Krivine J-P, Simmons R. (Eds.), Springer-Verlag, 1993.

[Thomas, 1996]
Thomas J.,*Vers l'intégration de l'apprentissage symbolique et de l'acquisition de connaissances basées sur les modèles: le système ENIGME*, Thèse de troisième cycle à l'Université Paris 6, décembre 1996.

[Thomas, Laublet et Ganascia, 1993]
Thomas J., Laublet P., Ganascia J-G, "A Machine Learning Tool for a Model-Based Knowledge Acquisition Approach", in *Proceedings of the 7th European Knowledge Acquisition For Knowledge-Based Systems Workshop (EKAW'93)*, p. 124-138, Aussenac N., Boy G., Linster M., Ganascia J.-G., Kodratoff Y., (Eds.), Lecture Notes in Artificial Intelligence 723, Springer Verlag, septembre 1993.

[Turner, 1993]
Turner R.M., "Context-sensitive Reasoning for Autonomous Agents and Cooperative Distributed Problem Solving", in *Proceedings of the IJCAI-93 Workshop on "Using Knowledge in its Context"*, p. 143-153, Brézillon P. (Ed.), Chambéry, 1993.

[Utgoff, 1989]
Utgoff P., "Incremental Induction of Decision Trees", *Machine Learning* **4**, 2, p. 161-186, 1989.

[Van de Velde, 1990]
Van de Velde W., "Reasoning, Behavior and Learning: A Knowledge Level Perspective", in *Cognitiva'90*, p. 451-465, Madrid, novembre 1990.

[Van de Velde, 1993]
Van de Velde W., "Issues in Level Modelling", In *Second Generation Expert Systems*, p. 211-231, David J-M, Krivine J-P, Simmons R. (Eds.), Springer-Verlag, 1993.

[van Harmelen et Bundy, 1988]
van Harmelen F., "Bundy A., Explanation Based Generalization = Partial Evaluation", *Artificial Intelligence* **36**, p. 401-412, 1988.

[van Someren, Zheng et Post, 1990]
van Someren M., Zheng L.L., Post W., Cases, "Models or Compiled Knowledge ? a Comparative Analysis and Proposed Integration", in *Current Trends in Knowledge Acquisition, Proceedings of the Fourth European Knowledge Acquisition Workshop (EKAW'90),* p. 339-355, Wielenga B., Boose J., Gaines B., Schreiber G., van Someren M. (Eds.), IOS Press, 1990.

[Varela, Thompson et Rosch, 1993]
Varela F., Thompson E., Rosch E., *L'inscription corporelle de l'esprit*, trad. fr.: Havelange V., Seuil, collection "La couleur des idées", Paris, 1993.

[Veloso, 1993]
Veloso M., "PRODIGY / ANALOGY: Analogical Reasoning in General Problem Solving", in Topics in Case-Based Reasoning, in *Proceedings of the First European Workshop on Case-Based Reasoning (EWCBR'93)*, p. 33-50, Wess S., Althoff K-D, Richter M. (Eds.), Lecture Notes in Artificial Intelligence 837, Springer-Verlag, 1993.

[Vera et Simon, 1993]
Vera A. H., Simon H. A., "Situated Action : A Symbolic Interpretation", *Cognitive Science* **17**, 1, janvier-mars 1993.

[Vrain, 1990]
Vrain C., "OGUST: A System that Learns Using Domain Properties expressed as Theorems", in *Machine Learning, An Artificial Intelligence Approach, Volume III,* p. 360-382, Kodratoff Y., Michalski R. (Eds.), Morgan Kaufmann, 1990.

[Weil-Barais, 1993]
Weil-Barais A. (Dir.), *L'homme cognitif,* Paris, Presses Universitaires de France, 1993.

[Wielenga et Breuker, 1986]
Wielenga B.J, Breuker J.A., "Models of Expertise", in *Proceedings of the Seventh European Conference on Artificial Intelligence (ECAI'86),* p. 306-318, Brighton, juillet 1986.

[Wielenga, Schreiber et Breuker, 1992]
Wielenga B.J., Schreiber A.T., Breuker J.A., "KADS: a modelling approach to knowledge engineering", *Knowledge Acquisition* **4**, p. 5-53, 1992.

[Wielenga, Van de Velde, Schreiber et Akkermans, 1993]
Wielenga B.J., Van de Velde W., Schreiber G., Akkermans H., "Towards an Unification of Knowledge Modelling Approaches", in *Second Generation Expert Systems*, p. 299-335, David J-M, Krivine J-P, Simmons R. (Eds.), Springer-Verlag, 1993.

[Wilkins, 1990]
Wilkins D.C., "Knowledge Base refinement as improving an incorrect and incomplete domain theory", in *Machine Learning, An Artificial Intelligence Approach, Volume III,* p. 493-513, Kodratoff Y., Michalski R. (Eds.), Morgan Kaufmann, 1990.

[Winograd, 1975]
Winograd T., "Frame Representations and the Declarative/Procedural Controversy", in *Representation and Understanding: Studies in Cognitive Science*, p. 185-210, Bobrow D.G., Collins A.M., Academic Press, 1975.

[Winograd et Flores, 1986]
Winograd T., Flores F., *Understanding Computers and Cognition : A New Foundation For Design*, Ablex, Norwood, NJ, 1986, trad. fr.: Peytavin J-L, *L'Intelligence Artificielle en question*, Paris, Presses Universitaires de France, collection “La politique éclatée”, 1989.

[Winston, 1975]
Winston P.H., "Learning Structural Descriptions From Examples", in *The Psychology of Computer Vision*, p. 157-209, Winston P.H. (Ed.), MacGraw-Hill, 1975.

[Winston, 1992]
Winston P.H., *Artificial Intelligence*, Third Edition, Addison-Wesley, 1992.

[Wogulis et Pazzani, 1993]
Wogulis J., Pazzani M., “A methodology for evaluating theory revision systems: Results with AUDREY II”, in *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence (IJCAI’93)*, Bajcsy R. (Ed.), p. 1128-1134, Chambéry, août 1993.

[Woods, 1970]
Woods W., "Transition network grammars for natural language analysis", *Communications of the ACM* **13**, 10, p. 591-606, 1970.

[Wrobel, 1996]
Wrobel S., "First Order Theory Refinement", in *Advances in Inductive Logic Programming*, p. 14-33, De Raedt L. (Ed.), IOS Press, 1996.

Annexe 1 : Le langage de la Mémoire de Travail

1) Langage des faits déclarés (description des actions):

Dans le champ “description d'une action”, tout ce qui est entre guillemets est un commentaire.

<déclaration> :: [<uneLigneDéclaration> ';']
<uneLigneDéclaration> :: <uneDéclarationDeTypeDAttribut>
| <uneDéclarationDeDomaineDAttribut>
| <uneDéclarationDeRelation>
<uneDéclarationDeTypeDAttribut> :: <unAttribut> <unEspace> <unType>
<uneDéclarationDeDomaineDAttribut> :: <uneConstante> | '(' <uneListeDeValeurs>')'
| <unAttribut> '.' '(' <uneListeDeValeurs> ')'
<uneDéclarationDeRelation> :: <unAttribut> <unEspace> <unAttribut>
<unAttribut> :: <uneConstante> | <unObjet> '.' <unAtome>
<unAtome> :: <uneChaîne> | <uneChaîne> '.' <unAtome>
<unType> :: 'ENTIER' | 'CHAINE' | 'REEL'
<uneListeDeValeurs> :: <uneValeur> | <uneListeDeValeurs> <unEspace> <uneValeur>
<uneConstante> :: <uneChaîne>
<uneValeur> :: <uneChaîne>
<unObjet> :: 'o' <unChiffre>

Remarque : les attributs sont monovalués.

2) Langage des faits renvoyés :

A la suite de l'exécution d'une action, EDINOS reçoit un ensemble de faits dans le format suivant. Ces faits sont stockés dans la MT.

<unFaitRenvoyé> :: <uneAffectation> | <unAttribut>
<uneAffectation> :: <unAttribut> <unEspace> <unAttribut>
<unAttribut> :: <uneChaîne> | <uneChaîne> '.' <unAttribut>

3) Langage des prémisses des règles de choix :

<unePrémisse> :: <uneFormule> <unConnecteurLogique> <uneFormule>
| <uneFormule>
<uneFormule> :: '(' <uneFormule> <unConnecteurLogique> <uneFormule> ')'
| <uneProposition>
<uneProposition> :: <unAttribut> <unOpérateurDeComparaison> <unAttribut>
| 'NON' '(' <unAttribut> ')' | <unAttribut>
<unAttribut> :: <uneChaîne> | <uneChaîne> '.' <unAttribut>
<unConnecteurLogique> :: 'ET' | 'OU'
<unOpérateurDeComparaison> :: '=' | '<' | '>' | '<=' | '>=' | '<>'

Exemple de prémisse :

(P1.P2.P3.P4 > Q1.Q2.Q3) ET (fait2.fait5 = 4 ET fait4 = 'V') ET fait1.fait6

Les attributs non comparés (<fait1.fait6> dans l'exemple ci-dessus), sont interprétés comme des valeurs booléennes (dans l'exemple, est-ce que <fait1> possède l'attribut <fait6> ?).

Quand l'attribut ou la relation n'existe pas, la proposition est classée fausse (ex: si <P3> n'est pas trouvée dans <P1.P2.P3.P4>, la prémisse de l'exemple ci-dessus est fausse).

De même, si un attribut "typé" n'est pas comparé, la proposition est classée fausse (ex: si <fait1.fait6> était typé, par exemple ayant été déclarée comme étant une chaîne, la formule ci-dessus serait fausse également).

4) Exemples :

Exemple 1:

Déclaration de l'action <infos>:

o1.ville CHAÎNE;
o1.âge ENTIER;
o1.sexe.(masculin feminin);

Contenu du champ “actions” :

evaluer: infos sur: enfant1

L'exécution de cette action pourra alors renvoyer par exemple les trois faits suivants:

enfant1.ville Villeurbanne
enfant1.âge 32
enfant1.sexe.masculin

Contenu du champ “règles de choix” :

si: enfant1.ville = 'Villeurbanne' alorsAller: n1;
si: (enfant1.âge > 25) ET enfant1.sexe.masculin alorsAller: n2;

Exemple 2:

Déclaration de l'action <caisseRattachement>:

o1.departement.CRAM CHAÎNE;
o2.departement o1.departement;

Contenu du champ “actions” :

evaluer: caisseRattachement sur: affilie et: conjoint

Faits renvoyés:

affilie.departement.CRAM Marseille
conjoint.departement affilie.departement

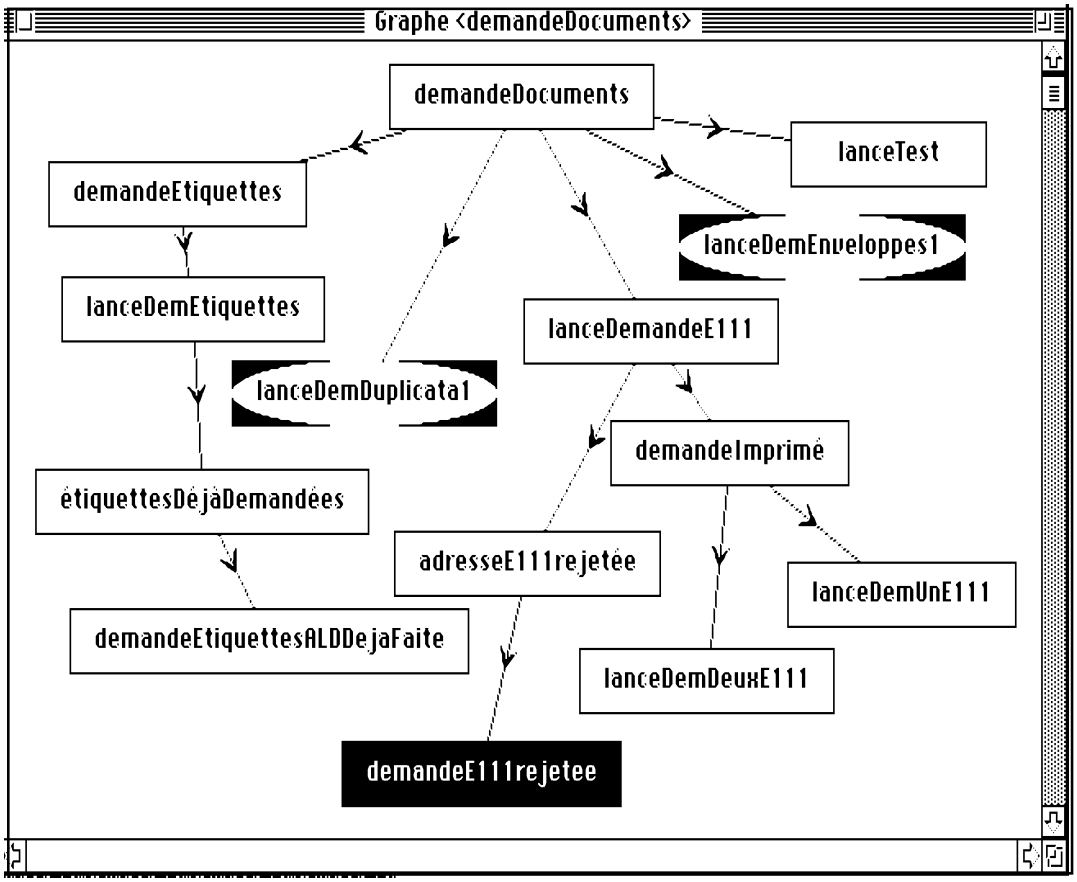
Contenu du champ “règles de choix” :

si: conjoint.departement.CRAM = 'Lyon' alorsAller: n1;

La règle ci-dessus ne sera pas activée pour cet exemple, puisque <conjoint.departement.CRAM> vaut <Marseille>.

Annexe 2 : Le graphe “demandeDocuments”

Cette annexe a pour objet de donner le contenu complet d'un graphe de nodules de situation.
Le graphe “demandeDocuments” a pour but d'enregistrer la demande de document du minitéliste (le minitéliste recevra ensuite par la poste le document désiré). Pour remplir cette tâche, le système peut être amené à donner des compléments d'informations au minitéliste, quant au contexte d'utilisation des documents (informations qui peuvent orienter le choix du document). Des contrôles peuvent aussi être effectués pour vérifier les droits du minitéliste-affilié relatifs au document demandé.



La figure ci-dessus donne un aperçu du graphe “demandeDocuments” : les branches des nodules“lanceDemDuplicata1” et “lanceDemEnveloppes1” ont été raccourcies afin que les autres puissent apparaître lisiblement. Le contenu complet du graphe est donné ci-dessous, dans le langage utilisé pour stocker la base sous une forme textuelle. Le graphe est lancé avec un paramètre (valant "enveloppes", "duplicata", "imprime", "etiquettes" ou "test"), qui est stocké dans la MT avant l'activation du nodule initial.

```
Graphe: demandeDocuments
\nod: demandeDocuments 236 @ 10
        \choix: si: enveloppes alorsAller: lanceDemEnveloppes1;
                si: duplicata alorsAller: lanceDemDuplicata1;
                si: imprime alorsAller: lanceDemandeE111;
                si: etiquettes alorsAller: demandeEtiquettes;
                si: test alorsAller: lanceTest
        \commentaires: paramètres d'appel:    enveloppes, duplicata, imprime,
                etiquettes ou test
        \finNod
\nod: demandeEtiquettes 28 @ 54
        \actions: demander: finEtiquettesALD;
        \finNod
\nod: demandeImprimé 371 @ 171
        \actions: demander: nbreExemplairesE111
        \choix: si: unE111 alorsAller: lanceDemUnE111;
                si: deuxE111 alorsAller: lanceDemDeuxE111;
        \finNod
\nod: lanceDemDuplicata2 303 @ 364
        \actions: evaluer: demandeDuplicata2;
        \finNod
\nod: demandeDuplicata 322 @ 175
        \actions: demander: confirmeDuplicata;
        \choix: si: confirme alorsAller: lanceDemDuplicata2;
        \finNod
\nod: demandeEtiquettesALDDejaFaite 41 @ 256
        \actions: evaluer: message
        \finNod
\nod: lanceDemDuplicata1 112 @ 148
        \actions: evaluer: demandeDuplicata1;
        \choix: si: plusieursDests alorsAller: demandeAdresseDuplicata;
                si: unSeulDest alorsAller: demandeDuplicata;
                si: dejaDemande alorsAller: duplicataDéjàDemandé
        \finNod
\nod: demandeAdresseEnvel 370 @ 232
        \actions: demander: confirmeAdresseEnvel
        \choix: si: adresseConfirmee alorsAller: lanceDemEnveloppes2;
                si: adresseRefusee alorsAller: adresseEnvelRejetée;
        \finNod
\nod: lanceDemEnveloppes2 233 @ 316
        \actions: evaluer: demandeEnveloppes2;
        \finNod
\nod: lanceDemEtiquettes 36 @ 106
        \actions: evaluer: demandeEtiquettes;
        \choix: si: dejaDemande alorsAller: étiquettesDéjàDemandées
        \finNod
\nod: étiquettesDéjàDemandées 16 @ 193
        \actions: demander: refusDemandeALD;
        \choix: si: message alorsAller: demandeEtiquettesALDDejaFaite
        \finNod
\nod: adresseDuplicataRejetée 5 @ 288
        \actions: demander: proposeJoindreMED;
        \choix: si: message alorsAller: demandeDuplicataRejetee
        \finNod
\nod: demandeEnveloppesDejaFaite 408 @ 270
        \actions: evaluer: message;
```

```
        \finNod
\nod: demandeAdresseDuplicata 0 @ 200
        \actions: demander: confirmeAdresseDupli;
        \choix: si: adresseConfirmee alorsAller: lanceDemDuplicata2;
                si: adresseRefusee alorsAller: adresseDuplicataRejetée;
        \finNod
\nod: lanceDemEnveloppes1 420 @ 87
        \actions: evaluer: demandeEnveloppes1
        \choix: si: plusieursDests alorsAller: demandeAdresseEnvel;
                si: unSeulDest alorsAller: demandeEnveloppes;
                si: dejaDemande alorsAller: enveloppesDéjàDemandées
        \finNod
\nod: demandeEnveloppes 327 @ 140
        \actions: demander: confirmeEnveloppes
        \choix: si: confirme alorsAller: lanceDemEnveloppes2
        \finNod
\nod: demandeDuplicataRejetee 20 @ 375
        \actions: evaluer: message;
        \finNod
\nod: adresseEnvelRejetée 427 @ 322
        \actions: demander: proposeJoindreGCOUR;
        \choix: si: message alorsAller: demandeEnveloppesRejetee;
        \finNod
\nod: demandeEnveloppesRejetee 398 @ 389
        \actions: evaluer: message;
        \finNod
\nod: enveloppesDéjàDemandées 441 @ 182
        \actions: demander: refusDemandeEnvel;
        \choix: si: message alorsAller: demandeEnveloppesDejaFaite
        \finNod
\nod: duplicataDéjàDemandé 191 @ 236
        \actions: demander: refusDemandeDuplicata;
        \choix: si: message alorsAller: demandeDuplicataDejaFaite
        \finNod
\nod: lanceDemDeuxE111 435 @ 319
        \actions: evaluer: demandeDeuxE111
        \finNod
\nod: lanceDemandeE111 261 @ 102
        \actions: evaluer: demandeImprimes;
        \choix: si: plusieursDests alorsAller: adresseE111rejetée;
                si: unSeulDest OU dejaDemande alorsAller: demandeImprimé;
        \commentaires: On ignore les cas ou les imprimés ont été déjà demandés :
                on va dans <demandeImprimé> quand même...
        \finNod
\nod: lanceTest 481 @ 36
        \actions: evaluer: testeMessagesRetour;
        \finNod
\nod: demandeDuplicataDejaFaite 171 @ 324
        \actions: evaluer: message;
        \finNod
\nod: demandeE111rejetee 226 @ 382
        \actions: evaluer: message
        \finNod
\nod: adresseE111rejetée 88 @ 221
        \actions: demander: proposeJoindreMED;
        \choix: si: message alorsAller: demandeE111rejetee
```

```
        \finNod
\nod: lanceDemUnE111 479 @ 235
        \actions: evaluer: demandeUnE111
        \finNod

\finGraphe
```

Annexe 3 : Statistiques sur le contenu de la BC

BC en service sur le serveur télématique à destination des affiliés
des Caisses de Prévoyance et de Retraite de la S.N.C.F.
au 1er octobre 1997.

Adresses utiles : base “adressesUtiles”
- 1 graphe
- 39 nodules
- 38 ecrans (38 demander)
- 1 declencher
- 3 niveaux

Demande de documents : base “demDoc”
- 1 graphe
- 28 nodules
- 12 ecrans (12 demander)
- 10 procédures (15 evaluer)
- 5 niveaux

Poser une question : base “correspondance”
- 6 graphes
- 70 nodules
- 48 ecrans (46 demander)
- 3 procédures (19 evaluer)
- 5 declencher
- 4 niveaux (max)

Bilan de santé :
base “bilanSanté”
- 1 graphe
- 123 nodules
- 112 ecrans (121 demander)
- 1 procédure (1 evaluer)
- 3 niveaux

base “demBilanSanté”
- 1 graphe
- 16 nodules
- 9 ecrans (11 demander)
- 4 procédures (6 evaluer)
- 1 declencher
- 7 niveaux

Dernières nouvelles de la CP : base “infosCPjuillet-97”
- 1 graphe
- 5 nodules
- 5 ecrans (5 demander)
- 2 niveaux

Dernières nouvelles de la CR : base “nouvellesAvril-97”
- 1 graphe
- 5 nodules
- 5 ecrans (5 demander)
- 2 niveaux

Commande d'un guide : base “commandeGuide”
- 1 graphe
- 1 nodule
- 2 ecrans (1 demander)
- 1 niveau

TOTAL :
- **13 graphes**
- **287 nodules**
- **235 ecrans (240 demander)**
- **17 procédures (41 evaluer)**
- **7 declencher**
- **6 procédures attachées**

Annexe 4 : Compte-rendu de la session de révision de l'exemple “ils se sont succédé”

[rapport donné par REVINOS]

ORIGINE DE LA REVISION :
auxiliaire?(auxiliaire),quelAuxiliaire?(etre),verbePronominal?(nonPronominal),accordAvecLeSujet.
BASE : participePassé1

[NP (verbeNonPronominal)]:
Créez la nouvelle procédure et
indiquez les faits qu'elle renvoie...

Ensuite seulement, cliquez sur 'Continuer'.

PARCOURS : REVINOS, informationManquante, nouvelleAction, nouveauNodule?, corrigeAction.

EN MOINS :
auxiliaire?(auxiliaire),quelAuxiliaire?(etre),verbePronominal?(nonPronominal),accordAvecLeSujet()

EN PLUS :
auxiliaire?(auxiliaire),quelAuxiliaire?(etre),verbePronominal?(nonPronominal),
fonctionDuPronomRéfléchi?(pronomReflechiCOD).
auxiliaire?(auxiliaire),quelAuxiliaire?(etre),verbePronominal?(nonPronominal),
fonctionDuPronomRéfléchi?(pronomReflechiCOI).

Poursuit avec l'exemple :
auxiliaire?(auxiliaire),quelAuxiliaire?(etre),verbePronominal?(nonPronominal),
fonctionDuPronomRéfléchi?(pronomReflechiCOD).
rejetteExemple

PARCOURS : REVINOS, ajoûterAction, reprendreUneActionTerminale, nouveauNodule?, corrigeGraphe, corrigerGraphe, accrocheAprès, corrigeRègles, corrigerRègles, créationDuneRègle, créeRègle.
---
corriger_règles (verbeNonPronominal, pronomReflechiCOD) : créeRègle.
ANCIENNES:

NOUVELLES :
si: pronomReflechiCOD alorsAller: pronomReflechiCOD;

EN MOINS :
auxiliaire?(auxiliaire),quelAuxiliaire?(etre),verbePronominal?(nonPronominal),
fonctionDuPronomRéfléchi?(pronomReflechiCOD).

EN PLUS :
auxiliaire?(auxiliaire),quelAuxiliaire?(etre),verbePronominal?(nonPronominal),
fonctionDuPronomRéfléchi?(pronomReflechiCOD),accordAvecLeSujet().

Poursuit avec l'exemple :
auxiliaire?(auxiliaire),quelAuxiliaire?(etre),verbePronominal?(nonPronominal),
fonctionDuPronomRéfléchi?(pronomReflechiCOI).
rejetteExemple

PARCOURS : REVINOS, ajoûterAction, reprendreUneActionTerminale, nouveauNodule?, corrigeGraphe, corrigerGraphe, accrocheAprès, corrigeRègles, corrigerRègles, créationDuneRègle, créeRègle.
---
corriger_règles (verbeNonPronominal, pronomReflechiCOI) : créeRègle.
ANCIENNES:
si: pronomReflechiCOD alorsAller: pronomReflechiCOD;

NOUVELLES :
si: pronomReflechiCOD alorsAller: pronomReflechiCOD;
si: pronomReflechiCOI alorsAller: pronomReflechiCOI;

EN MOINS :
auxiliaire?(auxiliaire),quelAuxiliaire?(etre),verbePronominal?(nonPronominal),
fonctionDuPronomRéfléchi?(pronomReflechiCOI).

EN PLUS :
auxiliaire?(auxiliaire),quelAuxiliaire?(etre),verbePronominal?(nonPronominal),
fonctionDuPronomRéfléchi?(pronomReflechiCOI),participePasséInvariable().

Poursuit avec l'exemple :
auxiliaire?(auxiliaire),quelAuxiliaire?(etre),verbePronominal?(nonPronominal),
fonctionDuPronomRéfléchi?(pronomReflechiCOD),accordAvecLeSujet().
accepteExemple

Poursuit avec l'exemple :
auxiliaire?(auxiliaire),quelAuxiliaire?(etre),verbePronominal?(nonPronominal),
fonctionDuPronomRéfléchi?(pronomReflechiCOI),participePasséInvariable().
accepteExemple

EXEMPLES VALIDÉS :
auxiliaire?(auxiliaire),quelAuxiliaire?(etre),verbePronominal?(nonPronominal),
fonctionDuPronomRéfléchi?(pronomReflechiCOD),accordAvecLeSujet().
auxiliaire?(auxiliaire),quelAuxiliaire?(etre),verbePronominal?(nonPronominal),
fonctionDuPronomRéfléchi?(pronomReflechiCOI),participePasséInvariable().

Annexe 5 : Compte-rendu de la session de révision de l'exemple des facilités de circulation (enfant malade)

[rapport donné par REVINOS]

ORIGINE DE LA REVISION :
benefFacCir(enfant),origine(enfant.origine.agent),celibataire(enfant.celibataire),
age>21ans(enfant.age>21ans),casParticulierEnfantDe21ans(enfant.malade),conditionsFacCir.
BASE : facCir2

A SOUMETTRE :
benefFacCir(enfant),origine(enfant.origine.agent),celibataire(enfant.celibataire),
age>21ans(enfant.age>21ans),casParticulierEnfantDe21ans(enfant.malade),conditionsFacCir.
benefFacCir(enfant),origine(enfant.origine.recueilli),confie(enfant.confié),
celibataire(enfant.celibataire),age>21ans(enfant.age>21ans),
casParticulierEnfantDe21ans(enfant.malade),conditionsFacCir.

Exemple non soumis :
benefFacCir(enfant),origine(enfant.origine.agent),celibataire(enfant.celibataire),
age>21ans(enfant.age>21ans),casParticulierEnfantDe21ans(enfant.malade),conditionsFacCir.
rejetteExemple

Exemple soumis :
benefFacCir(enfant),origine(enfant.origine.recueilli),confie(enfant.confié),
celibataire(enfant.celibataire),age>21ans(enfant.age>21ans),
casParticulierEnfantDe21ans(enfant.malade),conditionsFacCir.
rejetteExemple

PARCOURS : REVINOS, rectifierAction, reprendreUneActionTerminale, nouveauNodule?, corrigeGraphe,
corrigerGraphe, accrocheAuPère, corrigeRègles, corrigerRègles, substitutionDansUneRègle, modifieRègle.
---
corriger_règles (facCirEnfantDePlusDe21ans, donneFacCirCarteSiConditions75) : modifieRègle.
ANCIENNES :
si: enfant.origine.concubin ET (enfant.etudiant OU enfant.malade) alorsAller: donneFacCirSansCarte priorite: 1;
si: enfant.malade alorsAller: donneFacCirCarteSiConditions55 priorite: 2;
si: enfant.etudiant alorsAller: donneFacCirCarteSiConditions55 priorite: 2;
si: enfant.aCharge alorsAller: donneFacCirFamille;
si: enfant.auServiceNational alorsAller: donneFacCirSNA
NOUVELLES :
si: enfant.origine.concubin ET (enfant.etudiant OU enfant.malade) alorsAller: donneFacCirSansCarte priorite: 1;
si: enfant.malade alorsAller: donneFacCirCarteSiConditions75 priorite: 2;
si: enfant.etudiant alorsAller: donneFacCirCarteSiConditions55 priorite: 2;
si: enfant.aCharge alorsAller: donneFacCirFamille;
si: enfant.auServiceNational alorsAller: donneFacCirSNA

EN MOINS :
benefFacCir(enfant),origine(enfant.origine.agent),celibataire(enfant.celibataire),
age>21ans(enfant.age>21ans),casParticulierEnfantDe21ans(enfant.malade),conditionsFacCir().
benefFacCir(enfant),origine(enfant.origine.recueilli),confie(enfant.confié),
celibataire(enfant.celibataire),age>21ans(enfant.age>21ans),
casParticulierEnfantDe21ans(enfant.malade),conditionsFacCir().

EN PLUS :
benefFacCir(enfant),origine(enfant.origine.agent),celibataire(enfant.celibataire),
age>21ans(enfant.age>21ans),casParticulierEnfantDe21ans(enfant.malade),
conditionsFacCirEnfantMalade().
benefFacCir(enfant),origine(enfant.origine.recueilli),confie(enfant.confié),
celibataire(enfant.celibataire),age>21ans(enfant.age>21ans),
casParticulierEnfantDe21ans(enfant.malade),conditionsFacCirEnfantMalade().

Poursuit avec l'exemple :
benefFacCir(enfant),origine(enfant.origine.agent),celibataire(enfant.celibataire),
age>21ans(enfant.age>21ans),casParticulierEnfantDe21ans(enfant.malade),
conditionsFacCirEnfantMalade().
accepteExemple

Poursuit avec l'exemple :
benefFacCir(enfant),origine(enfant.origine.recueilli),confie(enfant.confié),
celibataire(enfant.celibataire),age>21ans(enfant.age>21ans),
casParticulierEnfantDe21ans(enfant.malade),conditionsFacCirEnfantMalade().
accepteExemple

Généralisation recherchée ? non

EXEMPLES VALIDÉS :
benefFacCir(enfant),origine(enfant.origine.agent),celibataire(enfant.celibataire),
age>21ans(enfant.age>21ans),casParticulierEnfantDe21ans(enfant.malade),
conditionsFacCirEnfantMalade().
benefFacCir(enfant),origine(enfant.origine.recueilli),confie(enfant.confié),
celibataire(enfant.celibataire),age>21ans(enfant.age>21ans),
casParticulierEnfantDe21ans(enfant.malade),conditionsFacCirEnfantMalade().

Table des figures

Un outil générique de conception et de révision coopérative de Bases de Connaissances s'appuyant sur la notion de situation

Ce travail s'inscrit dans la recherche en acquisition des connaissances et en apprentissage automatique pour la modélisation et la validation incrémentale de connaissances de résolution de problème.

Nous proposons un modèle simple de représentation des connaissances opératoires qui s'appuie sur la notion de situation, et présentons un outil de modélisation incrémentale et de révision coopérative pour les Bases de Connaissances (BC) exprimées dans cette représentation. Cet outil a été mis au point dans le cadre d'un projet de conception de dialogues télématiques personnalisés.

Dans notre modèle, chaque étape intermédiaire de résolution du problème est représentée explicitement dans le SBC sous la forme d'un objet simple et compréhensible appelé "nodule de situation". Les corrections et enrichissements de la BC sont effectués de manière incrémentale, c'est-à-dire au fur et à mesure de la découverte de cas mal résolus, et coopérative, c'est-à-dire en s'appuyant sur un utilisateur / concepteur de la BC compétent dans le domaine. Les caractéristiques de notre approche, que nous proposons de baptiser "révision située", sont les suivantes : l'objectif est de faire en sorte que le processus de révision de la BC soit facile pour l'utilisateur, basé sur des cas concrets, et opérant des corrections "prudentes" et validées.

L'outil REVINOS a été développé dans cette optique. Chaque phase de révision coopérative contient une étape de modélisation ou de réutilisation d'objets de la BC, à la charge du concepteur, puis une étape de correction proprement dite, effectuée de manière semi-automatique. REVINOS guide le concepteur tout au long du processus de révision et propose des généralisations à des cas concrets similaires. REVINOS offre l'originalité de chercher à valider les répercussions des corrections proposées, en soumettant au concepteur des exemples abstraits qui correspondent à des ensembles de cas concrets de résolution.

MOTS-CLÉS : Acquisition des Connaissances, Apprentissage automatique, Modélisation incrémentale des connaissances, Outil de révision coopérative, Notion de situation, Représentation des connaissances opératoires, Validation par simulation, Application à la conception de services télématiques.

A Knowledge Base Modelling and Cooperative Revision Tool based on the Concept of Situation.

This work takes place in Knowledge Acquisition and Machine Learning research for incremental modelling and validation of problem solving knowledge.

We introduce a simple model for representing procedural knowledge, based on the concept of situation, and we propose an incremental modelling and cooperative knowledge revision tool for Knowledge Bases (KB) expressed with that representation. This tool has been built and used for defining the sequence of French Minitel dialogues.

In our situation-based model, each step of problem solving is explicitly expressed in the KBS as a simple and understandable object called "situation nodule". The corrections and completions of the KB are incrementally done, i.e. as new ill-solved cases are discovered, and in a cooperative way, i.e. relying on the user, who is the KB designer competent in the domain. The goals of our approach, which we suggest to call "situated revision", are the following: the revision process should be easy for the user, based on concrete cases, and carrying out "cautious" and validated corrections.

The REVINOS tool has been developed with this perspective. Each revision step contains a KB objects modelling step, made by the designer, and then a semi-automatically correction step. REVINOS guides the designer during the whole revision process and suggests generalisations to similar concrete cases. One originality of REVINOS is to try to validate the repercussions of the suggested corrections, by submitting to the designer abstract examples that represent sets of concrete resolution cases.

KEYWORDS: Knowledge Acquisition, Machine Learning, Incremental Knowledge Modelling, Cooperative Revision Tool, Concept of Situation, Procedural Knowledge Representation, Validation by Simulation, Application to On-line Services Design.