INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA

UMI®
800-521-0600

HÉLÈNE VERRIÈRE

MODÉLISATION D'EXPLICATIONS DANS DES SYSTÈMES UTILISANT LE RAISONNEMENT PAR CAS

Mémoire
présenté
à la faculté des Études Supérieures
de l'Université Laval
pour l'obtention
du grade de maître ès sciences (M. Sc.)

Département d'Informatique
FACULTÉ DES SCIENCES ET DE GÉNIE
UNIVERSITÉ LAVAL

Novembre 1998

© Hélène Verrière, 1998

National Library of Canada
Acquisitions and Bibliographic Services
395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque nationale du Canada
Acquisitions et services bibliographiques
395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

Our file Notre référence

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

0-612-44986-6

Canada

RÉSUMÉ

Depuis toujours, et dans toute situation, l'homme éprouve le besoin de comprendre. Or, comment faire comprendre sinon expliquer. Le raisonnement par cas (RPC) est un type de raisonnement intuitif basé sur l'analogie. Il est donc a priori plus facile à comprendre que les autres types de raisonnement habituellement modélisés dans les systèmes à base de connaissances (SBC). Pourtant, les utilisateurs des SBC utilisant le RPC (SBC-RPC) peuvent aussi se heurter à certaines difficultés pendant la manipulation ou l'interprétation des résultats. Il est donc nécessaire d'étudier les types d'explications qui devraient être élaborés pour satisfaire les besoins des utilisateurs. L'étude des critères propres à la génération d'explications dans les SBC-RPC nous a permis d'élaborer un modèle générique d'explications. Il a ensuite été modélisé et analysé selon la méthodologie KADS afin de produire un prototype de SBC-RPC dans le domaine du jardinage en carrés doté d'un dispositif explicatif. Le résultat de notre recherche consiste en un modèle général réutilisable de l'élaboration des explications dans les SBC-RPC.

Nicole Tourigny
directrice de recherche

Hélène Verrière
étudiante graduée

AVANT-PROPOS

Pour l'aboutissement de ces deux années de travail dont le présent mémoire est le résultat, je désire remercier ma directrice de recherche Madame Nicole Tourigny, qui m'a toujours soutenue, encouragée et suivie dans l'avancement de mes travaux.

Je tiens également à saluer tous mes amis d'ici ou d'ailleurs, nouveaux ou anciens, qui m'ont accompagnée tout au long de cette expérience.

Un gros merci à mes sœurs pour m'avoir laissé partir, à mes beaux-frères et leurs enfants pour les avoir laissé partir pour venir me voir !

Finalement, je ne remercierai jamais assez mes parents qui m'ont permis de venir et qui malgré la distance, jugée trop grande, ont toujours été là à m'encourager, m'aider, me soutenir, me motiver, me féliciter et j'en passe, et qui, pendant toutes ces années d'études, ont toujours cru en moi.

TABLE DES MATIÈRES

LISTE DES FIGURES

CHAPITRE I

Introduction

Les systèmes à Base de Connaissances (SBC), souvent appelés Systèmes Experts(SE), font appel à des processus de résolution de nature heuristique plutôt qu'algorithmique [Gonzalez&Dankel 93]. Ils sont en outre caractérisés par les modes de raisonnement utilisés (déduction, induction, abduction, analogie, etc.), par le type de tâche effectuée (diagnostic, planification, classification, etc.), ainsi que par la façon dont les connaissances y sont représentées (règles, modèles, cas, objets, etc.) [Tourigny 1998]. La complexité croissante de ces systèmes et la diversité des domaines d'application ont rendu indispensable l'élaboration de dispositifs explicatifs qui en facilitent l'utilisation et la compréhension. En effet, le raisonnement modélisé peut être à l'origine des difficultés d'accessibilité de ces systèmes. C'est le cas du Raisonnement Par Cas (RPC), un type de raisonnement basé sur l'analogie. Les travaux effectués pour générer des explications dans les SBC utilisant le RPC (SBC-RPC) ne permettent pas, en général, de montrer la méthode de résolution suivie par ces systèmes et n'offrent donc pas de méthode suffisamment générale pour en élaborer. C'est pourquoi nous proposons un modèle générique d'explications et une méthode générale dont le but est de permettre aux concepteurs de SBC-RPC d'améliorer la qualité de leurs systèmes en les dotant de capacités explicatives.

Nous allons dans ce chapitre commencer par présenter la problématique des explications en général, ensuite nous entrerons plus précisément dans le sujet en présentant le problème de la génération d'explications dans les SBC-RPC. Les deux sections suivantes décriront les objectifs que nous avons choisis de poursuivre et les contributions que notre travail a apportées au domaine scientifique. Puis, nous présenterons la méthodologie que nous avons

suivie pour résoudre notre problème. Enfin, une introduction au prototype que nous avons développé montrera des résultats de notre travail. Finalement nous décrirons l'organisation de ce mémoire.

I.1 Pourquoi des explications

D'aussi loin que l'on puisse remonter, l'homme a toujours cherché, et ce même intuitivement, à comprendre les phénomènes qui l'entouraient. En effet, si l'on prend l'exemple des philosophes grecs, ceux-ci vouaient leur vie à parler des hommes, des éléments naturels et physiques qui composent le monde afin d'en mieux comprendre les origines, les fondements et le fonctionnement. Ainsi, même si la définition du dictionnaire [Robert 94] consiste à dire que "expliquer, c'est faire comprendre", nous en ajoutons une autre qui consisterait à dire que "expliquer, c'est comprendre" représente le principe fondamental qui permet à l'homme de progresser car il lui permet aussi d'apprendre. Ce besoin toujours présent de comprendre est à l'origine de toutes les grandes découvertes scientifiques ou autres. Ainsi, de chaque problème élucidé, en résulte souvent un nouveau, plus complexe, qui nécessite d'être expliqué à son tour. On peut très bien, pour illustrer ce principe, se référer au domaine de la médecine dans lequel chaque découverte d'un nouveau vaccin est suivie par celle d'un nouveau virus. Ce sont donc tous les grands scientifiques ou philosophes c'est-à-dire des gens avides de savoir qui ont permis à la science de progresser jusqu'à nos jours. Ceci rappelle une phrase de [Schank *et al.* 94] qui dit que "L'explication est au cœur de l'intelligence". En outre, ces quarante dernières années ont été marquées par une avancée des technologies inégalée jusqu'alors avec l'aérospatiale, le nucléaire mais aussi les moyens de communication et l'informatique. Cependant, si ce sont des hommes de sciences qui ont développé toutes ces nouvelles technologies, les utilisateurs eux, ne sont pas forcément préparés à les utiliser. En effet, étant donné l'expansion de l'informatique, il est difficile de nos jours d'échapper aux ordinateurs et ce, que ce soit au niveau professionnel ou privé. Les progrès réalisés ont pour conséquence que les logiciels développés deviennent très vite obsolètes de par la venue d'autres plus puissants, donc plus complexes, et pourtant sensés être plus adaptés et davantage conformes à la tâche à laquelle ils sont destinés. C'est pourquoi le logiciel ne peut remplir son rôle si l'utilisateur ne possède pas les connaissances nécessaires, tant au niveau informatique qu'au niveau du domaine dans lequel il exerce, pour en comprendre le fonctionnement tout comme les résultats obtenus, alors qu'une simple

explication aurait certainement permis à l'utilisateur de résoudre le problème auquel il était confronté. Ainsi, en Intelligence Artificielle (IA), l'utilisation des systèmes à base de connaissances (SBC) s'est très vite heurtée au scepticisme des utilisateurs [Moore&Swartout 88]. Par exemple, un système utilisant un raisonnement par inférence permet de déduire des conclusions suite à une série de règles déclenchées à partir de faits initiaux; seulement il arrive fréquemment que l'utilisateur ne connaisse pas la logique suivie, il peut donc aussi bien ne pas comprendre la conclusion elle-même. De nombreux travaux ont donc été réalisés afin de doter les SBC de capacités explicatives comme par exemple ceux de Buchanan et Shortliffe [Buchanan&Shortliffe 84] avec le système expert de diagnostic d'infections bactériennes MYCIN ou Gréboval [Gréboval 92] avec le projet SATIN (Système d'Aide au Traitement des Infections Néonatales). En effet, les premiers systèmes se limitaient souvent à générer des explications en paraphrasant les règles déclenchées, celles-ci n'étaient donc pas toujours exactes à cause d'un manque de connaissances ou encore semblaient peu naturelles ou même compréhensibles aux utilisateurs peu familiers avec la terminologie ou les concepts utilisés [Swartout&Moore 93]. De plus, les SBC portent sur un vaste domaine faisant appel à l'étude de plusieurs types de raisonnement. Le Raisonnement Par Cas (RPC) est un type de raisonnement particulier, basé sur l'analogie, qui utilise la similarité entre expériences pour résoudre de nouveaux problèmes [Kolodner 93; Riesbeck&Shank 89; Watson 97]. Cependant, même si ce type de raisonnement paraît élémentaire, il n'en est pas toujours de même pour l'utilisateur qui reçoit une solution du système [Goel *et al* 96b]. Que représente cette solution ? Est-elle adaptée au problème traité ? Que faut-il en faire ? Ces questions apparemment assez simples peuvent se poser à l'utilisateur, de même que de nombreuses autres. Le système doit alors être doté de capacités explicatives afin de répondre aux interrogations relatives à ses résultats. C'est le problème que nous avons choisi d'étudier pour notre travail de maîtrise et qui va être décrit dans la section suivante.

I.2 Problème

Les systèmes qui nous intéressent sont les Systèmes à Base de Connaissances utilisant le Raisonnement Par Cas (SBC-RPC). Connaissant les problèmes auxquels sont confrontés les utilisateurs de tels systèmes, notre problème consiste donc à déterminer comment élaborer des dispositifs explicatifs propres aux SBC-RPC qui pourraient augmenter la confiance des utilisateurs et rendre ces logiciels plus accessibles. Les dispositifs explicatifs existants sont

généralement propres à des systèmes particuliers en proposant un type spécifique d'explications [Muñoz-Avila&Hüllen 96], pour une tâche précise [Goel *et al.* 97] et n'offrent donc que peu de support qui puisse être réutilisé pour un autre système. Les méthodes existantes ne permettent pas non plus d'évaluer de manière générale la nature des besoins en explications dans de tels systèmes et ne renseignent donc pas sur la manière de les réaliser. De plus, les dispositifs explicatifs sont généralement utilisés en tant qu'outil interne au système pour améliorer ses propres performances comme par exemple dans les travaux de [Macedo *et al.* 96; Bento *et al.* 94b] et ne sont donc pas destinés aux utilisateurs finaux. Ils ne permettent donc pas de définir les types d'explications qui seraient adaptés aux besoins des utilisateurs. En effet, comment peut-on être sûr de l'interprétation qui sera faite des résultats ? Quel est le rôle du raisonnement par cas dans celle-ci ? Quelle différence fait-il avec les autres types de raisonnement ? Autant de questions sans réponses qui nous ont poussée à étudier le problème général de l'élaboration d'explications dans les SBC-RPC, notre hypothèse de travail étant que des explications sont nécessaires dans le cadre des SBC, comme l'ont d'ailleurs souligné de nombreux chercheurs dont [David *et al.* 93].

I.3 Objectifs

L'objectif général de notre travail de maîtrise consistait en l'élaboration d'une méthode de génération d'explications dans les SBC-RPC. Les explications étant généralement spécifiques au domaine traité [Schank *et al.* 94; Greer *et al.* 94], nous avons voulu placer notre travail dans un cadre général dont le résultat constituerait une base à laquelle se référer lorsqu'on veut produire des explications dans des SBC-RPC. Voici les objectifs spécifiques visés par ce travail :

- Élaborer un modèle générique qui contienne l'ensemble des critères spécifiques à la génération d'explications.
- Explorer les besoins en explications des utilisateurs dans les systèmes à base de connaissances utilisant le raisonnement par cas.
- Générer des explications qui soient proches de l'utilisateur en tant que fonctionnalité réalisée pour celui-ci.
- Tester la méthode de développement KADS pour modéliser les explications définies dans le modèle générique.
- Développer un prototype mettant en application les modèles proposés.

- Élaborer une méthode suffisamment générique qui puisse servir de support lors de la génération d'explications dans d'autres systèmes.

I.4 Contribution du mémoire

Notre travail de maîtrise sur la génération des explications se distingue des autres recherches effectuées dans ce domaine par les contributions suivantes. Tout d'abord, nous avons réalisé un modèle générique d'explications contenant l'ensemble des critères à prendre en compte lors de l'élaboration d'un dispositif explicatif, alors que la plupart des travaux ne font que considérer un voire deux de ces critères en fonction de leurs besoins et ne considèrent pas de manière spécifique et détaillée les autres. Ensuite, nous avons procédé à la mise en œuvre de tous les types d'explications du modèle générique, en tenant compte de tous les critères propres à la génération d'explications, dans un même prototype alors que souvent seuls un ou deux types sont considérés. De plus, nous avons innové en faisant l'étude des explications dans les SBC-RPC d'une manière différente. En effet, nous avons utilisé la méthode KADS pour modéliser nos explications dans des SBC-RPC et nous avons élaboré des explications destinées aux utilisateurs du système en fonction de leur nature et non pas des explications destinées à être utilisées par le système pour améliorer ses capacités de résolution de problème ou d'apprentissage. Ce faisant, nous avons élaboré un modèle d'expertise propre aux explications. Finalement, nous avons défini une méthode pour générer des explications dans des SBC-RPC qui peut servir de base à l'élaboration de tout module d'explications, et dans la littérature, ce genre de méthode, basée sur un modèle générique d'explications, utilisant la méthode KADS et appliquée au RPC n'existe pas à notre connaissance.

I.5 Méthodologie utilisée

Pour atteindre nos objectifs, notre première étape consistait à faire une étude des travaux réalisés sur les explications dans les systèmes à base de connaissances (SBC) tout d'abord, puis dans les SBC-RPC ensuite, afin d'extraire les critères importants pour générer des explications. L'issue de cette recherche était d'élaborer un modèle générique d'explications contenant l'ensemble des critères ou caractéristiques spécifiques à l'élaboration des explications. À partir de ce modèle, nous avons pu concevoir une application de SBC-RPC dans le domaine du jardinage en carrés dans laquelle nous avons exploré et imaginé l'ensemble des besoins en explications qu'un utilisateur pourrait éprouver. Le choix de cette

application a été guidé par la facilité d'accès du domaine pour lequel nous disposions d'informations, mais surtout par le fait qu'il fait appel à plusieurs tâches de résolution de problèmes, dont la classification de légumes potagers et la configuration de jardins en carrés nous permettant de disposer d'un environnement riche pour la génération d'explications. Le domaine d'application ainsi créé et le modèle générique nous ont alors permis de faire l'analyse, selon la méthodologie de développement de logiciels KADS, des explications dans les SBC-RPC. Celle-ci devait permettre de valider le modèle générique d'explications et d'obtenir une représentation des explications sous la forme de modèles structurés et à un niveau d'abstraction suffisamment élevé pour assurer la robustesse et la réutilisabilité du modèle. Finalement, afin de vérifier la validité de la méthode nous avons réalisé un prototype, EDEN, qui est l'implémentation de l'application de jardinage en carrés et du module d'explications modélisés selon KADS.

I.6 Le système (générateur conçu pour Élaborer Des Explications à partir d'un modèle géNérique) : des exemples d'explications

EDEN est le prototype qui permet d'illustrer une partie des résultats de notre travail de maîtrise. Il est doté d'un dispositif explicatif qui est destiné aux utilisateurs et qui permet d'expliquer la classification et le domaine spécifique des légumes potagers dans une application de raisonnement par cas. Ainsi les capacités d'EDEN permettent de :

- Expliquer le domaine d'application :

Qu'est ce que la classification de légumes potagers ?

- Expliquer les principes du raisonnement utilisé :

Qu'est ce que le raisonnement par cas?

- Expliquer les connaissances du domaine des légumes potagers:

Que signifie le terme Liliacées ?

Quelles sont les propriétés potagères de la carotte ?

- Expliquer la classification obtenue :

Pourquoi le légume décrit est-il classé dans la famille des Chénopodiacées ?

- Expliquer le raisonnement suivi :

Comment le système raisonne-t-il pour classer les légumes ?

- Expliquer pourquoi une solution a été préférée à une autre :

Pourquoi la classe trouvée est Chénopdiacées et pas Cucurbitacées ?

On trouvera des exemples concrets d'explications dans le chapitre VII dont la section 2 montre une illustration des capacités d'EDEN et des explications qu'il peut générer.

I.7 Plan du mémoire

Le mémoire est composé de huit chapitres. Le premier chapitre présente la problématique de la génération d'explications dans les systèmes à base de connaissances et plus particulièrement dans ceux utilisant le raisonnement par cas, les objectifs que nous voulons atteindre et les solutions que nous proposons. Le chapitre deux permet d'introduire l'environnement des systèmes à base de connaissances utilisant le raisonnement par cas. Le chapitre trois décrit les travaux qui ont été réalisés dans le domaine des explications dans ces systèmes et les principes de la méthodologie de développement KADS. Le chapitre quatre présente le modèle générique d'explications que nous avons élaboré. Les chapitres cinq et six consistent en la modélisation des explications par la méthode KADS du modèle d'explications dans le cadre des SBC-RPC en décrivant les phases d'analyse et de conception. Le chapitre sept décrit le prototype EDEN que nous avons réalisé et montre ses fonctionnalités par un exemple d'illustration. Une discussion sur les résultats obtenus clôt ce chapitre avant de terminer le mémoire par une conclusion au chapitre huit.

CHAPITRE II

Les SBC-RPC : des Systèmes à Base de Connaissances (SBC) utilisant le Raisonnement Par Cas (RPC)

L'objectif de ce chapitre est de décrire les Systèmes à Base de Connaissances utilisant le Raisonnement Par Cas (SBC-RPC) afin de bien situer le contexte dans lequel nous avons élaboré un modèle générique d'explications et implémenté un prototype EDEN. Après avoir rappelé les causes de l'échec de nombreux Systèmes à Base de Connaissances dits de première génération (SBC1), nous décrirons ce qu'il fut convenu de nommer les Systèmes à Base de Connaissances de deuxième génération (SBC2), et dont le développement repose sur un changement de paradigme, celui de la modélisation explicite des connaissances à différents niveaux [David 95]. En particulier Newell [Newell 82] a distingué les niveaux abstraits de connaissances *(knowledge level)* et le niveau des symboles ou de l'implémentation *(symbol level)*. Par la suite, nous détaillerons les SBC faisant appel à une forme particulière d'analogie, le Raisonnement Par Cas (RPC). Au cours de ce chapitre, nous insisterons sur l'importance de produire des explications dans les SBC en général, et en particulier dans les SBC-RPC.

II.1 Les Systèmes à Base de Connaissances (SBC)

En Intelligence Artificielle (IA), discipline visant " l'étude des moyens d'effectuer à l'aide d'ordinateurs, certaines des activités dites intelligentes de l'être humain" [Luger&Stubblefield 93; Simian&Tourigny 96], l'étude des SBC constitue une branche particulière qui a suscité l'intérêt de multiples chercheurs du domaine depuis de nombreuses années. En effet, les SBC,

tentent de résoudre des problèmes souvent réservés aux humains, par des techniques prises à l'IA, afin de fournir des solutions de la qualité d'un expert. Leur but peut aussi consister essentiellement à aider des experts ou des utilisateurs dans leurs travaux de résolution de problèmes. Les SBC se distinguent des systèmes informatiques classiques en premier lieu par le type des connaissances qu'ils utilisent, mais aussi par d'autres caractéristiques que nous allons décrire dans les paragraphes suivants.

II.1.1 Quelques aspects historiques : l'échec des SBC dits de première génération (SBC1)

Depuis leur origine, située dans les années 60, les SBC ont beaucoup évolué[1]. La première génération de SBC, ou SBC1, n'avait pas pour but de modéliser le raisonnement humain, mais d'exploiter les connaissances spécifiques d'un domaine, obtenues par le biais d'un expert, avec une méthode de résolution appropriée pour acquérir les performances de résolution de problème d'un niveau expert [Luger&Stubblefield 98]. C'est le cas de DENDRAL, un des premiers SBC, développé à la fin des années 60 à Stanford. L'un des objectifs à l'origine du développement de tels systèmes était de parvenir à centraliser l'expertise. Souvent, les experts d'un même domaine sont dispersés géographiquement et leur expertise n'est accessible que localement, quand elle n'est pas perdue. Aussi, réunir toute celle-ci permet d'enrichir l'expérience et d'aider le travail des experts qui peuvent alors consulter un ensemble de problèmes résolus avant de réaliser leur propre résolution de problème. En l'insérant dans ce qui constitue la base de connaissances, l'expertise n'est alors pas perdue et est donc, à long terme, réutilisable. Le processus d'acquisition des connaissances vise à identifier et à obtenir les connaissances qui permettront aux SBC de fournir des solutions de qualité au moins équivalente à celles d'un expert du domaine. Dans le développement des SBC1, l'acquisition des connaissances, encore considérée comme un goulot d'étranglement [David 95] était vue comme une activité de transfert des connaissances entre l'expert et le SBC. Elle consistait généralement en une suite d'entrevues entre un expert du domaine et un ingénieur de la connaissance, au cours desquelles l'expert décrivait ses connaissances du domaine et ses méthodes de résolution de problème. Ensuite, l'ingénieur de la connaissance était chargé d'implémenter cette connaissance dans un système de manière à le rendre

---

[1] Dans cette section, nous nous inspirerons en particulier des travaux de [David 95; Luger&Stubblefield 93; Simian&Tourigny 96]

efficace et "intelligent" dans son comportement. Un des premiers problèmes que l'on peut évoquer concernant les SBC1 réside dans le fait que les experts ont généralement des difficultés à reproduire sous forme orale leur expertise dans les moindres détails, les connaissances étant souvent incomplètes ou incertaines. Ensuite, les connaissances étaient représentées de manière uniforme sous forme de règles de production, et concernaient aussi bien les tâches à réaliser et le mode de réalisation de ces tâches que les connaissances du domaine ou les méthodes sous-jacentes, limitant ainsi les possibilités du système à résoudre des problèmes complexes. De plus, les données du problème et les éléments de la solution associée n'étaient pas séparés. Tout ceci se traduisait par un problème de réutilisabilité des systèmes [David 95]. En effet, le manque d'abstraction et le faible niveau de spécificité des connaissances du domaine ajoutés au fait que la connaissance et les méthodes de résolution n'étaient pas séparées ne conféraient aux systèmes aucune modularité ni aucune généralité pour permettre de réutiliser l'expertise acquise dans d'autres systèmes ou d'identifier les structures récurrentes existantes. En outre, les systèmes ainsi réalisés montraient des problèmes de robustesse liés principalement à trois facteurs. Tout d'abord, si la connaissance d'un expert évolue constamment et ce de manière naturelle dans le temps, celle d'un système informatique est entièrement dépendante de la connaissance qu'on lui incorpore. Aussi, les SBC n'étaient-ils pas "conscients" de leurs limites en matière d'expertise et ne pouvaient donc reconnaître leur incapacité à résoudre un problème entraînant ainsi des erreurs lors de la résolution de problème. De plus, à la différence des humains, les SBC1 ne pouvaient adapter les méthodes de résolution de problème ou les connaissances à utiliser en fonction de la complexité du problème. Enfin, même si la représentation des connaissances utilisée était sous forme de règles qui doivent en théorie être indépendantes les unes des autres, le fait d'ajouter ou de modifier des règles, du fait de la dépendance des connaissances entre elles, pouvait modifier le comportement général du système. Enfin, un dernier handicap que l'on peut associer aux SBC1 provient de la difficulté à comprendre les solutions trouvées pour des utilisateurs expérimentés ou non. C'est pourquoi, est apparue très tôt la nécessité de doter les SBC de dispositifs explicatifs pour permettre aux experts et aux utilisateurs de tels systèmes de comprendre les résultats, les démarches suivies ainsi que les connaissances manipulées. Malheureusement, l'acquisition des connaissances, dont l'objectif était le transfert des connaissances de l'expert à un modèle d'exécution dans un contexte méthodologique de prototypage rapide, ne permettait pas en particulier de distinguer les types de connaissances

utilisées, par exemple celles du domaine et celles du contrôle du SBC. Les connaissances informatives disponibles dans les SBC se sont le plus souvent avérées insuffisantes pour produire des explications utiles et adaptées à l'utilisateur. Ce sont toutes ces limites, à savoir les difficultés associées à la phase d'acquisition des connaissances, une réutilisabilité difficile, le manque de robustesse et des explications peu satisfaisantes, qui ont amené les chercheurs à réaliser d'autres travaux afin d'élaborer une nouvelle génération de SBC appelée systèmes experts de seconde génération ou SBC2 [David 95].

II.1.2 Caractéristiques des SBC2

Comme nous venons de le décrire, l'échec de nombreux SBC1 est expliqué notamment par leur mode uniforme de représentation, l'inexistence d'alternatives dans les modes de résolution de problèmes et le faible niveau d'abstraction de la représentation des connaissances. L'efficacité et la pertinence de l'élaboration des SBC reposent maintenant sur un ensemble de critères que nous allons décrire ci-après. Ceux-ci ont été déterminés dans la perspective de pallier les limites des SBC1, et en fonction des objectifs associés aux SBC. Certaines de ces caractéristiques sont en particulier développées dans [David 95; Gonzalez&Dankel 93; Luger&Stubblefield 98].

- **La modélisation explicite des connaissances utilisées** doit permettre de construire des systèmes qui soient à la fois compréhensibles (explications), maîtrisables et maintenables (robustesse). En effet, contrairement aux systèmes conventionnels dans lesquels la connaissance est de type algorithmique, dans les SBC, elle est notamment de type heuristique. Aussi, il est nécessaire d'utiliser fortement les connaissances spécifiques du domaine propres aux experts. En effet, dans de telles applications, l'expérience et le savoir-faire des experts associés à une formation théorique permettent davantage de résoudre des problèmes qu'une connaissance standard issue directement des livres. L'acquisition des connaissances explicites et la représentation du comportement du système à un meilleur niveau d'abstraction permettent de générer un plus grand nombre d'explications qui peuvent alors, si la connaissance nécessaire est disponible, être adaptées en fonction du niveau de détail désiré. De plus, l'abstraction apporte une certaine généralité ou généricité, propriété fondamentale pour fournir des explications.
- **La prise en compte de différents modèles du domaine et de diverses méthodes de résolution** a pour objectif de résoudre différents types de problèmes, de niveaux variés de

complexité, utilisant de multiples connaissances. Le système peut ainsi s'adapter en fonction du problème. Cette caractéristique permet de fournir au système une capacité de raisonner plus proche de celle utilisée par un expert humain qui combine inconsciemment plusieurs méthodes de résolution afin de parvenir à des résultats. Les systèmes deviennent plus riches et plus robustes, car ils peuvent accomplir davantage de tâches de résolution. Ainsi, les SBC ne sont plus uniquement des systèmes à base de règles, mais d'autres raisonnements peuvent être modélisés comme le raisonnement à base de modèles, les réseaux neuronaux ou encore le raisonnement par cas, raisonnement qui nous intéresse ici.

- **La distinction entre les connaissances à utiliser et la manière de les implémenter** signifie que la connaissance doit être modélisée à un certain niveau d'abstraction (*knowledge level*, [Newell 82]) distinct du langage d'implémentation (*symbol level* [Newell 82]). Ceci permet de décrire le processus de résolution de problème à un meilleur niveau d'abstraction [David 95]. Ainsi, un SBC doit être composé d'une Base de Connaissances (BC) qui contient la connaissance théorique et pratique de l'expert et d'un Moteur d'Inférence (MI) qui utilise le contenu de la BC pour résoudre les problèmes posés. La séparation des deux types de connaissances permet de générer des explications plus adaptées aux besoins car elles peuvent avoir à fournir de l'information à un niveau des connaissances sans avoir à utiliser la connaissance de résolution de problème. De plus, cette particularité permet de produire des SBC plus ouverts. En effet, ils deviennent modulaires et on peut ainsi plus facilement exporter une partie d'un système dans une autre application sans avoir à modifier l'ensemble du système. On peut alors réutiliser indifféremment soit les connaissances relatives au domaine car elles sont contenues dans la BC, soit les techniques de résolution, etc.
- **L'utilisation d'une méthode de conception appropriée** permet de concevoir des SBC en respect avec les objectifs et les besoins des experts et des utilisateurs. En effet, la conception des SBC se fait selon un cycle de vie composé d'un ensemble de phases hiérarchisées et structurées qui vont de la phase d'analyse des besoins en passant par la phase d'acquisition des connaissances et se terminant par une phase de validation du système créé. La méthode utilisée dans notre cas est KADS (Knowledge Acquisition and Design Structuring) qui sera décrite en section III.2. Cette méthode est basée sur une approche de modélisation explicite des connaissances.
- **La génération d'explications doit être considérée comme une tâche de résolution à part entière.** En effet, la difficulté à produire des explications satisfaisantes dans les SBC1 a mené

les chercheurs à considérer cette tâche non plus comme un traitement auxiliaire facultatif, mais comme une tâche spécifique qui méritait sa propre analyse et ses propres processus de résolution. Nous détaillerons les caractéristiques associées à la génération d'explications dans les Chapitre III et Chapitre IV.

Cependant malgré les efforts entrepris, les SBC2 montrent encore un certain nombre de limites [Luger&Stubblefield 98; Swartout&Moore 93; David 95]. En effet, même si les connaissances sont maintenant modélisées de manière plus explicite, la compétence d'un SBC reste toujours limitée à son domaine d'expertise [Luger&Stubblefield 98], ce qui continue à nuire à la robustesse du système. La phase d'acquisition des connaissances demeure une étape délicate qui dépend autant de la complexité du domaine que du niveau d'expertise que l'on veut atteindre. En effet, la connaissance profonde du domaine reste difficile à capturer, car d'un point de vue expert, elle est évidente et constitue la base de son raisonnement. Pourtant, c'est cette connaissance qui aide à élaborer des explications mieux adaptées aux utilisateurs qui ne possèdent pas toujours les fondements théoriques pour comprendre le raisonnement suivi et les choix stratégiques entrepris. Les SBC étant maintenant développés dans des domaines d'application très diversifiés avec des problèmes à résoudre de complexité croissante, la vérification de leurs compétences s'avère être une tâche difficile à accomplir. L'étape de validation est donc incertaine. Enfin, une fois terminé, un système informatique, comme nous l'avons déjà mentionné, ne peut évoluer sans un apport humain, aussi, la capacité d'apprentissage demeure-t-elle un défi. Nous verrons par la suite que la plupart de ces limites peuvent être non pas forcément supprimées, mais du moins amoindries grâce à l'utilisation d'un type de raisonnement adapté et d'une technique de modélisation approfondie.

II.1.3 Architecture des SBC2

Le prototype EDEN que nous avons développé, et qui sera décrit plus loin dans ce mémoire, constituant un type particulier de SBC, il nous paraît important de donner les éléments de base de l'architecture des SBC (Figure 1). Celle-ci concerne les SBC en général, et le plus souvent à base de règles, aussi cette architecture va-t-elle évoluer au cours de notre analyse, le raisonnement appliqué étant le raisonnement par cas. De plus, notre étude portant sur les dispositifs explicatifs, le module d'explications sera par la suite complété.

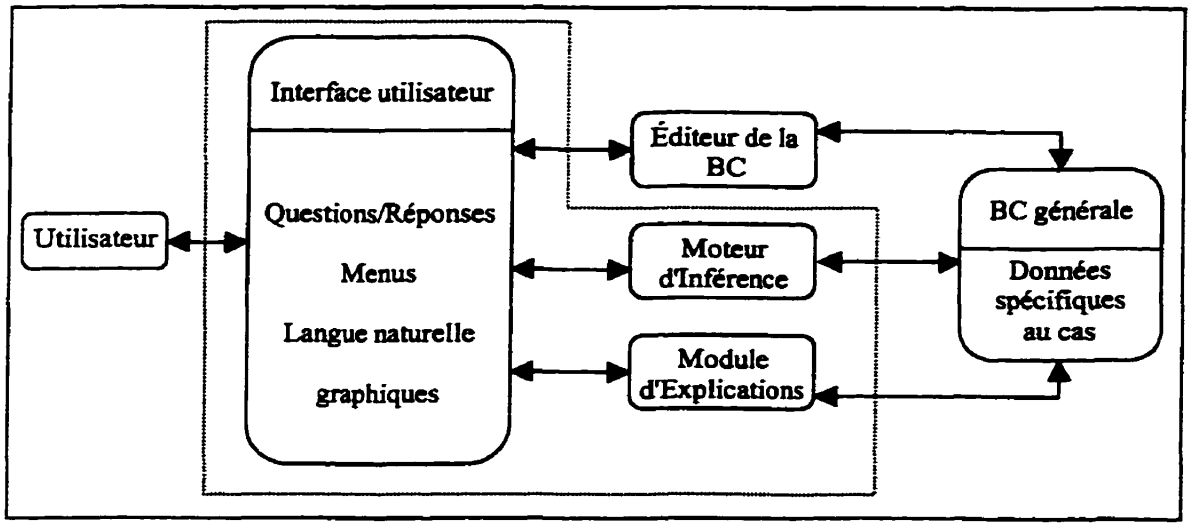

Figure 1. Architecture des SBC [Luger&Stubblefield 98]

Les composantes décrites ci-après constituent les SBC.

- **L'interface utilisateur** permet de gérer l'interaction entre le SBC et l'utilisateur. Elle doit répondre aux besoins des utilisateurs et adopter un mode de communication convenable (menus, graphismes, discours en langage naturel, ...) en fonction du problème posé.
- Lors de la résolution d'un problème spécifique, toutes les données relatives au cas traité doivent être conservées pour une utilisation future probable dans ce que l'on appelle les **données spécifiques au cas**. Elles servent beaucoup pour la génération d'explications. En effet, elles contiennent les justifications associées aux choix réalisés, les mesures de confiance, les résultats obtenus, les connaissances impliquées générales ou non, etc. Par exemple, en conservant toutes les étapes suivies au cours d'une résolution de problème spécifique, le module d'explications peut générer une trace du raisonnement permettant d'expliquer à l'utilisateur les résultats obtenus.
- Le **module d'explications** a pour rôle de générer des explications plausibles, adaptées au système afin de fournir une aide aux utilisateurs. Plusieurs types d'explications peuvent être générés dépendamment de la situation tels que des explications sur les connaissances du domaine ou sur le raisonnement suivi par le système par exemple. Ce module utilise les connaissances spécifiques du cas pour élaborer des explications appropriées à partir de techniques explicatives implémentées par les concepteurs en fonction des besoins des utilisateurs.

- L'**éditeur de la BC** permet au concepteur de mettre à jour la BC. Il peut accéder au module explicatif afin de détecter les erreurs de programmation ou de conception. Il permet ainsi d'améliorer les performances du système.
- La **base de connaissances générales** (BC) contient la connaissance associée à la résolution de problème de l'application. Cette connaissance peut être représentée sous différentes formes dont les règles de production dans le cas de systèmes à base de règles, les schémas (frames), les cas, etc.
- Le **moteur d'inférence** (MI) interagit avec la BC et les données du cas spécifiques pour résoudre un problème posé. Grâce à ses mécanismes de raisonnement, il permet d'inférer de nouveaux faits qui sont alors ajoutés à la BC.

Nous venons donc de décrire les constituants principaux que doit avoir de manière générale un SBC afin de réaliser les tâches spécifiées, par exemple la classification, le diagnostic, la planification. Comme nous l'avons énoncé précédemment, l'une des raisons qui ont poussé les chercheurs du domaine à reconsidérer les SBC1 était la nécessité de doter ces systèmes de capacités explicatives. Nous allons, dans la section suivante décrire plus précisément les besoins en explications que les SBC rencontrent en général.

II.1.4 Explications dans les SBC2

La génération d'explications représente un domaine complexe, car très vaste. En effet, dans la réalité, lorsqu'une explication est produite, elle correspond à un schéma spécifique lié à une situation précise. C'est l'étude de cette situation et des facteurs entrant en jeu qui détermine le contenu d'une explication. Et c'est précisément cette phase d'analyse qui constitue le noyau de la tâche explicative. Nous allons ici décrire les besoins en explications des SBC d'une part, et nommer quelques approches, ensuite, nous énumérerons les critères d'évaluation des explications. En effet, ce sont ces critères qui nous permettront, à l'issue de notre travail, de juger de la validité du modèle que nous avons implémenté.

II.1.4.1 Besoins en explications

La nécessité de doter les SBC de capacités explicatives est donc, comme nous l'avons déjà mentionné, reconnue depuis longtemps. Il est apparu très vite que pour rendre les SBC utiles et accessibles, il fallait guider et aider leurs utilisateurs à en comprendre le fonctionnement, la logique. En effet, même si les SBC arrivent à remplir leur rôle i.e. à accomplir leur tâche avec

les mêmes résultats qu'un expert, seuls les experts peuvent a priori juger de l'exactitude de ces résultats. Ainsi, quand un système médical diagnostique pour un patient une maladie, il est important pour le docteur d'être sûr de la pertinence du diagnostic posé avant d'administrer au patient le traitement associé. Aussi, si ce même docteur émet des doutes ou désire de l'information supplémentaire, il est nécessaire de doter le système d'un dispositif spécifique qui lui permettra de répondre autre chose que "l'ordinateur a dit ça !". Dans la réalité, le patient exigerait du docteur qui a dressé le diagnostic une justification précise des éléments (symptômes identifiés, résultats d'analyses sanguines, par exemple) et du raisonnement qui l'ont amené à cette conclusion. Nous venons d'énoncer que la motivation des concepteurs de SBC dans la génération d'explications est de rendre leur système accessible à leurs utilisateurs. Aussi, la génération d'explications repose avant tout sur l'étude des besoins des utilisateurs. L'un des premiers besoins en explications concerne donc la mise en confiance des utilisateurs dans les résultats trouvés par un SBC. Par là, on entend qu'elles doivent le persuader de la pertinence et de la performance du système d'une part, et de l'autre clarifier son fonctionnement et ses recommandations. Persuader un utilisateur qu'un résultat est correct et faire comprendre ce résultat ne sont pas deux tâches similaires. Un niveau d'abstraction les distingue. Ce niveau peut paraître un détail de perception. Cependant, c'est l'objectif des SBC2 de pouvoir générer des explications qui satisfassent le niveau de perception des utilisateurs et ce en fonction de leur propre niveau de connaissance ou des exigences qu'ils ont à l'encontre du système. Cette représentation des besoins de l'utilisateur en fonction de ses compétences constitue le modèle de l'utilisateur. Ainsi, les explications vont devoir tenir le rôle d'outil de persuasion, de formation, de correction d'erreur du logiciel. En effet, souvent, les explications sont nécessaires pour montrer les limites d'un logiciel, en expliquant son raisonnement, l'expert peut alors localiser et identifier l'erreur. Il faut donc avant toute chose identifier les tâches explicatives ou buts d'explications [Bourcier *et al.* 94] que l'on veut implémenter dans un système. Ceci implique qu'il faut considérer la tâche explicative comme une tâche de conception à part entière [Bourcier *et al.* 94], et que celle-ci doit posséder ses propres techniques, méthodes et stratégies de résolution [Swartout&Moore 93]. Les concepteurs des SBC1 considéraient que les connaissances contenues dans la BC étaient suffisantes pour générer des explications compréhensibles. Or, expliquer un événement ou un fait consiste généralement à faire appel à sa propre expérience ou à des connaissances autres que théoriques pour rendre son discours accessible. Ainsi

d'après Swartout et Moore [Swartout&Moore 93] : "expliquer quelque chose à un utilisateur nécessite la construction d'une histoire qui raconte les choses que l'utilisateur ne comprend pas en faits, plans et buts qu'il connaît et accepte". C'est pourquoi, il a été reconnu que la connaissance contenue dans les SBC n'était en général pas suffisante pour permettre de générer des explications plausibles. Ceci explique l'échec des SBC1 à expliquer leurs résultats. En effet, les premiers systèmes à disposer d'un module explicatif, comme MYCIN, un SBC de diagnostic d'infection bactérienne créé par Shortliffe en 1976, ne fournissaient qu'une trace du raisonnement sous la forme d'un paraphrasage des règles déclenchées. Cette approche, par sa simplicité, offre quelques avantages. En effet, elle ne nécessite pas beaucoup d'effort une fois que le système est réalisé. De plus elle permet aux personnes familières avec l'utilisation des règles de production de suivre pas à pas le raisonnement suivi. Le problème concerne les autres utilisateurs, ceux qui veulent comprendre le diagnostic dans sa généralité. En effet, le fait de paraphraser chaque règle lorsqu'elle est déclenchée permet de comprendre les règles individuellement, mais pas l'approche suivie globalement. Cette constatation est en contradiction avec l'objectif initial qui consiste pour la plupart des SBC à former d'autres experts et à leur faire gagner du temps en profitant de l'expertise déjà existante. Si la connaissance profonde permettant de comprendre les fondements du raisonnement n'existe pas, le système échoue. D'autres approches spécifiques des SBC1 consistent à élaborer des textes prédéfinis ou des textes à trous qui sont instanciés lors de la demande d'explications de l'utilisateur. Ces techniques sont utiles pour représenter et expliquer les connaissances du domaine en terme de concepts et de relations. C'est ce que l'on appelle les connaissances statiques et les explications statiques associées. Celles-ci dépendent de la bonne maintenance du système car si des modifications sont apportées, à moins d'une mise à jour manuelle, elles peuvent conduire à de fausses explications à l'utilisateur. L'ensemble de ces explications, appelées explications uniformes, même en langage naturel, semblait peu naturel et incomplet. De ces limites est donc venu le besoin de représenter les connaissances du système différemment afin de pouvoir les rendre accessibles pour générer des explications [Swartout&Moore 93]. D'autres approches ont été développées pour générer des explications dans les SBC2. Celles-ci consistent notamment à raisonner et à représenter la connaissance à un meilleur niveau d'abstraction comme dans le système AIDE développé par Bourcier [Bourcier *et al.* 94; Kassel 93]. D'autres travaux ont pour objectif de produire des explications réactives comme le générateur de Moore [Moore 89]. Les explications y sont construites sur la

base d'un dialogue avec l'utilisateur en tenant compte de ses réactions concernant les résultats fournis. Cet aspect rétroactif permet de construire des explications dynamiques, c'est-à-dire celles qui concernent le processus de résolution, et elles sont élaborées pendant la réalisation de la tâche. Les explications reconstructives [Wick&Thomson 92] reposent, quant à elles, sur une théorie selon laquelle pour générer de bonnes explications il ne suffit pas d'adopter un processus de cause à effet, généralement utilisé dans les systèmes à base de règles car facile à implémenter, mais à partir d'une reconstruction du raisonnement suivi. Ces explications permettent d'utiliser plusieurs types de raisonnement. En effet, il s'agit de combiner un raisonnement logique qui permet de résoudre le problème avec un raisonnement abductif qui consiste à utiliser des hypothèses qui permettent de construire le scénario causal [David 95].

Nous venons de décrire plusieurs approches qui ont permis de produire des explications de diverses façons. L'évaluation des explications produites par un système constitue un autre aspect important à considérer lors du développement de SBC.

II.1.4.2 Évaluation des explications

Plusieurs approches, et certaines ont été citées au paragraphe précédent, ont été suivies par les chercheurs concernant la génération d'explications. En fait, on ne peut trouver dans la littérature une étude globale permettant de dresser un cadre générique qui pourrait être réutilisable par n'importe quel concepteur qui veut implémenter dans son système un dispositif explicatif. En effet, on peut remarquer que les travaux sur le sujet ne sont pas aussi nombreux que ceux existants sur les SBC eux-mêmes par exemple. Ainsi comme le souligne Dhaliwal dans [Dhaliwal 98], la question "comment développer un dispositif explicatif ?" n'a jamais rencontré le même succès que la question "comment développer un système expert ?", et ce, même si la génération d'explications est reconnue comme étant un élément indispensable au succès des SBC2. En réalité, les travaux réalisés suivent deux directions : soit ils réalisent une revue de littérature sur les travaux réalisés jusqu'ici en faisant l'analyse des points forts et des inconvénients de chacune des méthodes[1], soit ils décrivent une méthode spécifique développée dans le cadre d'un projet bien précis. De la même manière, les

---

[1] Parmi ceux de la première catégorie, mentionnons [Tourigny 94]

Parmi ceux de la deuxième catégorie, mentionnons [Leake 95]

techniques d'évaluation des explications restent purement descriptives. L'évaluation des explications au travers des travaux réalisés est effectuée en accord avec un certain nombre de critères. Ces critères, même s'ils concernent des points et des qualités spécifiques des explications, ne font pas l'objet d'analyses aboutissant sur l'implémentation d'un module permettant l'évaluation de dispositifs explicatifs. Ainsi, et ceci est représentatif du domaine des explications en général, pour évaluer les explications qu'un SBC produit, il faut vérifier que le système répond à un ensemble de propriétés. Il faut noter que la diversité des objectifs des SBC, les utilisateurs à qui ils sont destinés et la complexité des domaines auxquels ils s'adressent rendent difficile le fait qu'un système réponde à toutes les exigences. En effet, certaines propriétés des explications ne sont pas forcément appropriées à la situation. Les critères sont encore une fois relatifs aux besoins des utilisateurs et au système.

Cependant, d'après Swartout [Swartout&Moore 93], et de manière générale, la qualité d'un dispositif explicatif repose sur un ensemble de cinq facteurs :

- **la fidélité** : les explications doivent reproduire exactement le raisonnement du système. Elles doivent être en accord avec les décisions et les motivations du système. Pour cela il faut, comme nous l'avons déjà mentionné, qu'elles soient basées sur la même connaissance profonde que celle de la résolution de problème;
- **la compréhensibilité** : les explications pour être utiles doivent être compréhensibles. Ce critère dépend d'un certain nombre de facteurs comme la terminologie utilisée, le niveau d'abstraction auquel se situent les explications par rapport au raisonnement du système, la prise en compte des perspectives des utilisateurs, les qualités linguistiques des explications et enfin la rétroaction avec l'utilisateur. La prise en compte de ces facteurs dépend bien sûr de la nature des explications que l'on veut produire et du contexte même de l'application;
- **la suffisance** : un système doit être en mesure de produire des explications lui permettant de répondre aux diverses questions des utilisateurs. Pour cela il est nécessaire d'étudier les besoins en explications du système;
- **l'efficacité** : la production d'explications ne doit pas nuire à l'efficacité du système et sa rapidité d'exécution;
- **une élaboration de complexité limitée** : l'élaboration d'explications ne doit pas augmenter de manière inconsidérée la complexité de conception du système.

D'autres méthodes ne portent pas sur l'évaluation des explications elles-mêmes, mais sur l'évaluation de l'utilisation des explications. On peut en effet considérer que si des explications ne sont pas utilisées, c'est qu'elles ne correspondent pas aux besoins des utilisateurs, et donc qu'elles ne sont pas adaptées. On réalise alors leur évaluation. Ainsi, dans [Dhaliwal 98], les explications sont évaluées en fonction de leur forme et de leur type. Tout d'abord, l'interface qui représente un moyen de communication étroit entre le système et l'utilisateur doit être conviviale. Un effort doit donc être fait sur le plan ergonomique. Les connaissances doivent être représentées d'une manière adaptée au niveau d'expertise de l'utilisateur. Ensuite, il apparaît que l'utilisation des explications repose sur une relation étroite entre le type de l'explication produite et la catégorie ou niveau de compétence de l'utilisateur. Chaque catégorie d'utilisateurs possède ses propres exigences quant à la nature des explications qu'il désire. Si on fournit uniquement des explications sur les connaissances du domaine à un expert, celui-ci ne sera pas satisfait et inversement. L'évaluation des explications est donc déterminée à partir des besoins initiaux d'une application. Si ces besoins sont satisfaits, les explications s'avèrent en général suffisantes.

Comme nous l'avons mentionné précédemment, les SBC2 se distinguent des SBC1 par les différents types de raisonnement et les modes de représentation des connaissances qu'ils utilisent. Ainsi, un raisonnement inductif comme celui utilisé dans les Systèmes à Base de Règles (SBR) ne sera pas expliqué de la même manière que celui d'un système utilisant le RPC car ils ne suscitent pas les mêmes interrogations aux utilisateurs. De même les connaissances impliquées ne sont pas représentées sous le même formalisme, donc elles ne seront pas utilisées de la même manière pour générer des explications. Les SBC auxquels nous nous intéressons étant les SBC utilisant le RPC nous allons, dans la section suivante, décrire ce type particulier de raisonnement.

II.2 Le Raisonnement Par Cas (RPC)

Les SBC2 ont la particularité de pouvoir modéliser différents types de raisonnement en fonction de l'application développée. Dans certaines situations, il n'est pas nécessaire, ou même possible, de modéliser un raisonnement pour générer une solution. Dans certains cas, l'utilisation d'expériences antérieures peut s'avérer utile pour gagner du temps et éviter de recalculer une solution si celle-ci a déjà été trouvée. Le raisonnement par cas utilise cette

stratégie de résolution. Nous allons dans ce qui suit, reprendre plus précisément ses principes et ses caractéristiques. Ensuite, nous étudierons, comme dans le cas des SBC les besoins en explications des SBC utilisant le RPC.

II.2.1 Introduction au RPC

L'idée du raisonnement par cas n'est pas nouvelle. En effet, intuitivement tout le monde utilise le raisonnement par cas. Prenons un exemple. Chaque automne, quand les premiers froids commencent à se faire sentir, des milliers d'oies du Canada partent du Nord et font une halte au Cap Tourmente avant de poursuivre leur vol vers des régions plus chaudes telles que les États Américains. Au printemps, dès que la chaleur commence à revenir, ces mêmes oies repassent par le Cap Tourmente le temps de quelques jours pour rejoindre leur point de départ. Cette situation est la même chez tous les oiseaux migrateurs. Ce phénomène surprenant, n'est toutefois pas à attribuer à un niveau d'intelligence supérieur. Un processus systématique, quoique inconscient, pousse ces oiseaux tous les ans à la même période à identifier un ensemble de facteurs, qui réunis, leur permet de reconnaître une situation particulière. Celle-ci correspond, pour ces oies, à un départ de l'endroit d'où elles se trouvent car les conditions ne sont plus favorables. Le chemin qu'elles empruntent n'est pas tracé, il fait lui aussi partie du plan qu'elles ont reconnu et tout le trajet est guidé par le contenu de ce plan. En bref, on pourrait dire que les oies se "rappellent" et qu'elles réutilisent le même schéma quand la situation se présente. Cependant, certaines années le froid est précoce, elles adaptent alors leur plan en faisant un départ anticipé. Cet exemple reflète bien le principe de base du RPC. Il s'agit d'un raisonnement basé sur l'analogie et purement intuitif car nous n'en avons pas généralement conscience dans notre vie quotidienne. En effet, chaque jour nous sommes confrontés à des situations qui nous rappellent des situations similaires déjà vécues par le passé. Évidemment, certains facteurs peuvent varier d'une situation à l'autre, nous nous arrangeons donc, et ce toujours de manière inconsciente, pour réutiliser l'information de notre expérience passée qui nous est utile et adapter les éléments manquants ou incomplets en fonction de la situation présente. Le schéma se répète ainsi indéfiniment et ce de manière naturelle.

Les chercheurs en sciences cognitives ont été les premiers à s'intéresser à la modélisation du RPC, mais ce sont Schank et Abelson qui sont à l'origine du RPC en 1977 [Watson 97]. Leur théorie était basée sur le fait que la connaissance générale des situations que nous possédons

dans notre cerveau est en fait enregistrée sous forme de scripts qui nous permettent de dresser des attentes et de réaliser des inférences [Schank 82]. Il suffit donc de pouvoir représenter la connaissance contenue dans ces scripts avec les problèmes et les solutions pour pouvoir élaborer des SBC, et ce, même si on ne possède pas de modèle explicite de la connaissance [Watson 97]. Schank est donc le premier à avoir élaboré un modèle cognitif pour le RPC. Mais, ce n'est qu'en 1983 que Janet Kolodner va construire le premier système RPC, CYRUS, qui est une implémentation du modèle de la mémoire dynamique de Schank. Depuis, de nombreux systèmes RPC ont été développés de même que des outils de développement de tels systèmes. En effet, ne plus avoir à modéliser systématiquement un raisonnement entier pour produire des solutions, mais plutôt chercher à déterminer si un problème similaire n'a pas été déjà résolu, et alors réutiliser sa solution en l'adaptant au contexte représente un atout majeur. Une telle approche permet non seulement de gagner du temps, mais aussi de développer des applications dans des domaines d'application où la connaissance est entièrement heuristique. Il permet en particulier de trouver des solutions à des problèmes malgré des données manquantes ou incertaines. En effet, certains domaines se caractérisent par leur manque de données concrètes, absolues. Par exemple, le RPC est très utile dans le domaine du diagnostic médical [Watson 97]. En effet, il arrive que des situations inattendues, ne répondant à aucun schéma classique, se produisent. Il est alors très intéressant de pouvoir vérifier si une telle situation ne s'est pas déjà produite et de voir quels avaient été les traitements administrés avant d'essayer plusieurs diagnostics dans le but d'en trouver un qui soit cohérent. Évidemment, on peut se retrouver avec une situation inédite, mais peut-être en se rappelant d'autres situations qui avaient en commun au moins un des symptômes observés, de nouvelles hypothèses peuvent être émises. C'est pourquoi, de nombreux travaux réalisés sur le RPC combinent ce dernier avec d'autres types de raisonnement tels que le raisonnement abductif par exemple. Les cas retrouvés servent de point de départ pour émettre un ensemble d'hypothèses qui seront par la suite vérifiées. Si elles s'avèrent cohérentes, on les retient pour trouver la solution. De plus, dans le passé, une erreur de diagnostic a pu être commise, mais ensuite réparée. Si une trace de cette erreur est conservée dans le cas, quand celui-ci est retrouvé au cours d'une nouvelle résolution de problème, l'erreur précédemment commise est alors rappelée et expliquée, évitant ainsi qu'elle soit reproduite. Le RPC présente donc un certain nombre d'avantages pour la production d'explications. En effet, un cas par lui-même constitue une explication. Il contient

les données du problème et la solution associée alors qu'une explication représente souvent une relation causale entre des données et des solutions.

En résumé, nous pouvons dire que le RPC permet de résoudre une variété de tâches. Ainsi, il peut adapter et combiner des solutions antérieures pour résoudre un nouveau problème, expliquer de nouvelles situations en fonction de celles déjà survenues ayant une même expérience, critiquer de nouvelles situations par rapport à des cas antérieurs, raisonner sur des cas précédents pour comprendre une nouvelle situation, ou encore construire une solution généralisée basée sur des cas antérieurs [López de Mántaras&Plaza 97]. Aussi, maintenant que nous avons défini le RPC, nous allons dans le paragraphe suivant décrire les caractéristiques des systèmes l'utilisant. Nous préciserons également les différentes tâches du RPC qui nous intéressent.

II.2.2 Caractéristiques des SBC-RPC

Comme nous l'avons précédemment mentionné, les SBC-RPC sont des SBC qui modélisent un type particulier de raisonnement analogique qui est le RPC. Ainsi, l'architecture globale d'un SBC-RPC est constituée essentiellement des mêmes éléments de base que celle d'un SBC. La seule différence réside dans le contenu des divers composants. Ainsi, les connaissances sont représentées sous forme de cas et la Base de Connaissances est appelée Base de Cas. Ensuite le moteur d'inférence est remplacé par ce que l'on appelle le module du RPC, composé de quatre processus qui constituent le cycle du RPC. Nous allons, dans ce qui suit, décrire la Base de Cas et le cycle du RPC.

II.2.2.1 Base de Cas

Même si un cas peut contenir des données représentables dans une Base de Données (BD) classique, la principale différence existant entre les bases de cas et les BD réside dans la nature de l'information utilisée. Une BD exige des données précises pour retrouver efficacement des enregistrements, alors qu'une base de cas peut utiliser des connaissances incertaines et incomplètes pour sa résolution de problèmes. En effet, le RPC est surtout utilisé dans des contextes où la connaissance n'est pas modélisée ou est difficile à l'être. Ainsi, on peut considérer qu'un cas représente une histoire précise avec un énoncé, un dénouement et les incidents survenus pendant le passage de l'un à l'autre. Watson dans [Watson 97] définit un cas comme une pièce contextualisée de la connaissance représentant une expérience qui contient la leçon retenue, le contenu du cas, et le contexte dans lequel la leçon peut être

utilisée. Le cas peut être un compte rendu d'un événement, d'une histoire, ou un certain enregistrement typique composé de l'espace du problème c'est-à-dire le problème qui décrit l'état du monde quand le cas s'est produit, et de l'espace de la solution, c'est-à-dire la solution qui établit la solution dérivée de ce problème. Cependant, le choix de l'information à représenter dans un cas ne suit pas de critères fixes. Elle est déterminée en fonction du type de l'application et des tâches que l'on veut résoudre. Ainsi, le système PROTOS [Kolodner 93] contient dans un seul modèle de cas la connaissance générale et la connaissance spécifique au cas. Ceci permet de disposer à tout moment de l'information spécifique au cas, utile pour générer des explications. Aussi, les cas sont généralement composés de deux types d'information : l'information indexée qui est utilisée lors du processus qui permet de retrouver les cas similaires et l'information non indexée destinée à fournir un surplus d'information à l'utilisateur et notamment utilisée lors de la génération d'explications. Elle fournit à l'utilisateur de l'information contextuelle sur une valeur, mais elle n'est pas utilisée lors de la recherche de cas antérieurs. En principe, n'importe quel langage de représentation de la connaissance peut être utilisé pour les cas. Ceux-ci peuvent être représentés sous la forme de listes uniformes d'attribut-valeur, de graphes, de plans et de réseaux de contraintes, d'ensembles d'objets, etc. Le choix du langage de représentation doit être guidé par les méthodes de recherche de cas similaires et d'adaptation choisies. Par exemple, on ne peut pas appliquer des procédures d'assortiment de graphes à des listes d'attribut-valeur. Aussi, la qualité d'un bon système RPC est directement liée à la qualité de sa base de cas. C'est pourquoi celle-ci doit contenir un ensemble de cas qui soit représentatif du domaine traité. En effet, plus la base de cas est complète, plus l'expérience augmente et plus la pertinence des cas retrouvés est de qualité. Comme pour les SBC classiques, la difficulté dans les SBC-RPC concerne donc l'acquisition des connaissances. Il est nécessaire de représenter l'information pertinente de la description du problème dans un cas et ce à un bon niveau d'abstraction afin que les cas soient suffisamment généralisés pour être réutilisables dans le futur [Kolodner 93].

II.2.2.2 Cycle du RPC

Le fonctionnement du RPC est assez simple, il s'agit d'un raisonnement intuitif facile à comprendre et également facile à implémenter. Watson décrit les fonctionnalités qui caractérisent le RPC dans [Watson 97] comme suit :

- trouver dans la BC une description de problème qui soit la plus proche, ou encore un ensemble de problèmes similaires ;
- peut-être poser plusieurs questions pour confirmer le problème ou se concentrer sur le meilleur enregistrement ;
- offrir alors la solution connue du meilleur cas ;
- peut-être même adapter la solution retrouvée en fonction des différences entre le problème retrouvé et le problème courant.

Le RPC peut donc être défini selon un processus cyclique : un nouveau problème est comparé aux cas contenus dans la base de cas. Un ou plusieurs cas similaires sont alors retrouvés. La solution suggérée par le cas le plus similaire au cas problème est alors réutilisée et testée pour vérifier si elle est convenable. À moins que le cas retrouvé soit très similaire aux données du problème, la solution va être révisée produisant alors un nouveau cas qui peut être retenu. La figure 2 montre le cycle du RPC.

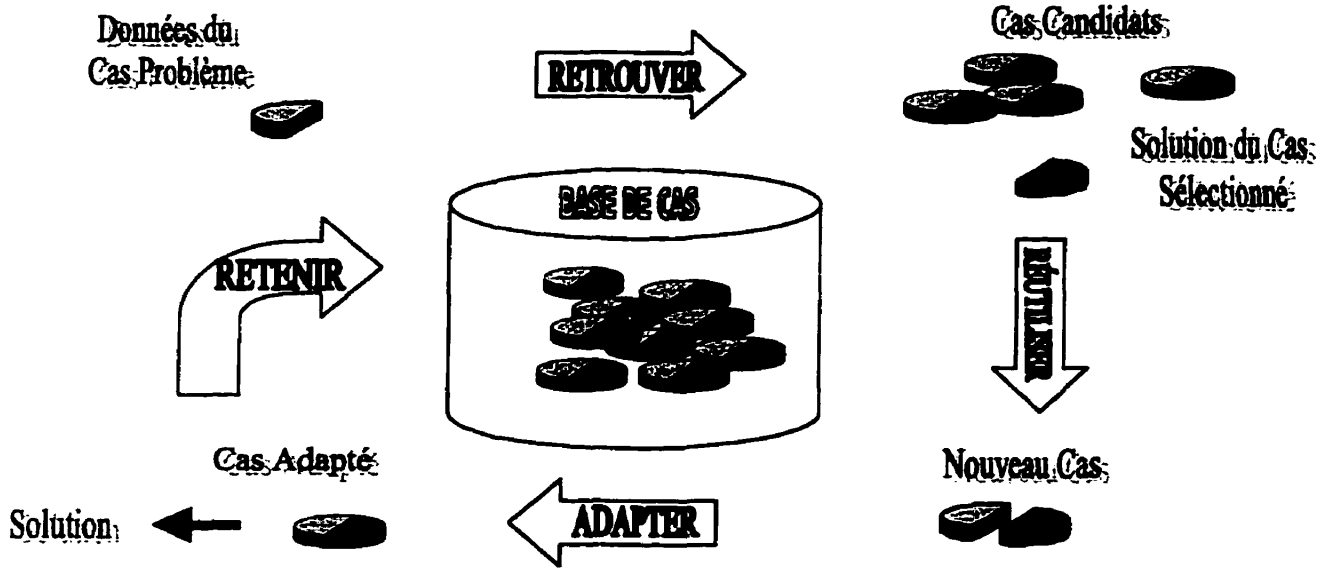

Figure 2. Cycle du RPC

Voyons maintenant plus en détail chacune de ces étapes.

1) **Retrouver** le(s) cas similaire(s)

Cette étape a pour objectif de retrouver dans la base de cas un cas déjà résolu qui correspond le plus possible aux données du problème courant. Le problème est donc décrit en fonction des descripteurs de problème. Ensuite, les cas sont recherchés en suivant les index. Finalement les cas les plus similaires sont retrouvés à l'aide de la fonction de similarité utilisée. Il existe plusieurs méthodes pour retrouver les cas, dont la méthode du plus proche

voisin et la méthode inductive [Watson 97; Kolodner 93; Capus&Tourigny 98b]. La technique du plus proche voisin consiste à calculer la somme des distances entre chaque attribut de cas problème et des meilleurs cas retrouvés. Le meilleur cas est celui qui obtient le plus petit score. On peut pondérer les attributs en fonction de leur importance. La technique inductive consiste à construire un arbre de décision qui classifie les cas de la base de cas. Comme techniques d'induction fréquemment utilisées, on peut citer ID3 [Quinlan 86], C4.5 [Quinlan 93], etc. Ensuite une recherche dans l'arbre en fonction des données du cas permet de retrouver les cas les plus similaires.

2) **Réutiliser** le cas pour tenter de résoudre le problème

Une fois qu'un ensemble de cas a été retrouvé, on sélectionne celui qui satisfait le mieux les exigences et on teste sa solution sur le cas courant. Très souvent, la solution étant satisfaisante, on la réutilise pour résoudre le problème et le processus se termine.

3) **Adapter** (Réviser) la solution proposée si nécessaire

Dans le cas où la solution retrouvée ne permet pas de résoudre entièrement le problème (voir figure 2), il est nécessaire de l'adapter. En effet, les données du problème pouvant différer, il est normal que la solution à apporter ne soit pas entièrement identique d'une situation à l'autre. De nos jours, le processus d'adaptation est celui qui demande le plus d'efforts. C'est pourquoi dans de nombreux systèmes l'adaptation est encore laissée aux soins des utilisateurs. Cependant, il existe plusieurs types de méthodes d'adaptation. Les deux principales sont l'adaptation par modification de la solution (ou générative) et l'adaptation par substitution heuristique (ou structurelle) [Gebhardt *et al.* 97]. La première est utilisée dans des domaines où la théorie est complète, elle est obtenue en réalisant la résolution de problème du début. Certains systèmes extraient et modifient la solution des cas sources, les autres utilisent l'information à propos de la dérivation de la solution à l'aide d'un modèle causal du domaine, soit comme une description des opérateurs avec leurs préconditions et leurs effets, ou une description des relations causales entre des symptômes et des diagnostics. L'adaptation heuristique réutilise des cas sources sans l'aide d'un outil en s'appuyant sur les heuristiques.

4) **Retenir** la nouvelle solution comme un nouveau cas

Cette dernière étape consiste à insérer le cas nouvellement résolu avec les données du problème et la solution associée dans la base de cas. Ceci permet de favoriser les capacités d'apprentissage du RPC dont la base s'enrichit continuellement au cours des résolutions.

Les 4 processus du RPC peuvent donc être clairement définis. Selon les systèmes et les applications chacune des parties est plus ou moins développée. En effet, le RPC permet de résoudre un ensemble de tâches variées, comme les SBC classiques. Nous allons décrire les principales dans la section suivante.

II.2.3 Les tâches des SBC-RPC

La "tâche classique" d'un système RPC consiste à trouver des cas antérieurs similaires qui aident ou permettent d'analyser de nouveaux problèmes ou de prendre de nouvelles décisions. [Weber 94]. Cependant, les SBC-RPC, comme tout SBC, sont utilisés pour résoudre un ensemble de tâches très variées. Ainsi, de manière générale, les SBC permettent de résoudre deux types de tâches : les tâches d'analyse et les tâches de synthèse [Watson 97; Gebhardt *et al.* 97]. Les premières consistent à classifier un objet dans une certaine catégorie en fonction d'un certain nombre d'attributs. On retrouve dans cette classe, les tâches de classification, de diagnostic, d'évaluation, de prédiction, de support de décision, etc. Les secondes, plus difficiles à implémenter, consistent à créer de nouvelles solutions en combinant des parties de solutions passées. Ce sont des tâches de planification, de conception ou de configuration. Elles nécessitent la prise en compte de nombreuses contraintes entre les éléments manipulés. Certains systèmes peuvent posséder les caractéristiques des deux tâches [Gebhardt *et al.* 97]. La figure 3 représente une classification de ces tâches telle qu'on peut la trouver pour tout SBC et notamment dans la bibliothèque des tâches génériques de KADS [Tansley&Hayball 93]. Néanmoins nous la montrons telle qu'elle a été réalisée par [Watson 97] pour les SBC-RPC.

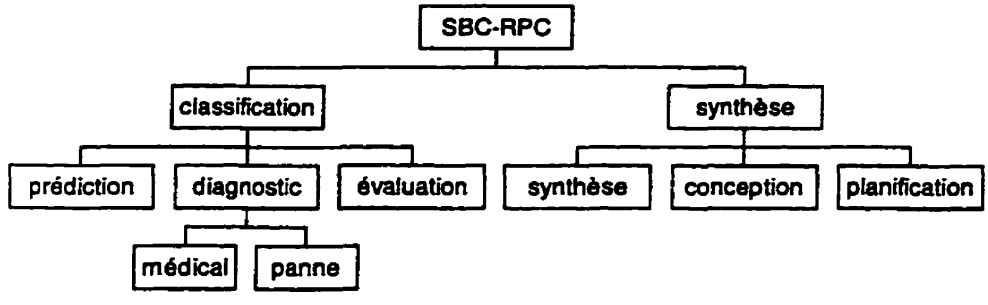

Figure 3. Les tâches des SBC-RPC [Watson 97]

Nous allons définir les tâches spécifiques de classification et de planification. En effet, ce sont les deux tâches de RPC modélisées dans l'application de jardinage en carrés que nous avons développée.

II.2.3.1 Une tâche d'analyse : la classification

L'objectif principal de la tâche d'analyse du RPC est d'accéder à certaines propriétés ou caractéristiques d'un nouveau problème, le cas courant, à la lumière de vieux épisodes, les cas sources [Gebhardt *et al.* 97]. La tâche de classification représente la fonction première des systèmes RPC. Elle consiste à se demander si une nouvelle instance ressemble suffisamment à une autre pour qu'on lui assigne la même classification. La classification est proche de la prédiction en cela que généralement un champ ou un attribut de chaque enregistrement va représenter le résultat, et la majorité des autres attributs de chaque cas servira à prédire le résultat [Watson 97]. Dans un SBC-RPC la base de cas est alors constituée de cas qui ont déjà été classifiés en accord avec un schéma uniforme ou hiérarchique. Étant donné un nouveau cas, le ou les cas les plus similaires sont retrouvés, et leur classe est adoptée si la similarité a atteint un seuil et si tous les cas voisins, de par leur description, sont en accord avec la classe. Si aucune classe ne correspond, un expert classifie le cas lui-même, et on obtient alors un nouveau cas qui est enregistré dans la base de cas [Watson 97]. Ainsi dans notre application de jardinage en carrés la tâche de classification consiste, à partir de la description d'un légume, à trouver la famille de légumes auquel il appartient. La tâche de classification dépend du processus qui permet de retrouver les cas. Celui-ci doit prendre en compte l'impact de chaque attribut du cas en y insérant des facteurs de poids. Il est nécessaire aussi, pour réaliser une bonne classification, de garder un certain niveau d'abstraction dans la définition des attributs demandés [Kolodner 93]. S'il s'agit d'un algorithme d'induction, celui-ci ne doit pas être trop exhaustif, il peut être utile de rajouter des informations à prendre en compte ou de réaliser une discrétisation des variables qui soit moins sélective. Pour des problèmes de classification, seul le processus RETROUVER est nécessaire. En effet, la phase d'adaptation n'est en général pas utile : si la classe du cas retrouvé ne convient pas, on choisit un autre cas qui convient mieux. D'après Gerbardt [Gebhardt *et al.* 97], les cas utilisés dans des tâches de classification n'ont pas besoin d'être adaptés. La tâche de classification est la tâche la plus couramment utilisée dans le RPC, de plus elle s'accorde bien à son cycle [Watson 97]. Elle est de plus relativement facile à réaliser contrairement aux tâches de synthèse qui nécessitent un plus gros effort.

II.2.3.2 Les tâches de synthèse

Tandis que la tâche d'analyse laisse le cas presque intact et lui attache seulement le résultat de l'analyse, la tâche de synthèse consiste à changer des parties du cas, à y ajouter des portions ou les deux. Les deux types de tâches diffèrent donc par leur besoin d'une étape d'adaptation. Essentiellement, un cas incomplet et inconsistant est donné et, avec l'aide de cas antérieurs, révisé ou complété avec de nouveaux éléments. Ainsi, dans le cas de l'application de jardinage en carrés, la tâche de synthèse consiste à retrouver un plan de jardinage en carrés en fonction d'une liste de légumes données, et à l'adapter en le modifiant pour qu'il satisfasse au mieux le contenu de cette liste en fonction des contraintes de l'application. Dans les tâches de synthèse, on retrouve en particulier, les tâches de planification, de conception, de diagnostic. Étant donné qu'il n'existe pas de frontière réelle entre elles, un système peut contenir les caractéristiques de plusieurs d'entre elles.

Lorsque la planification est utilisée dans des domaines où l'on réfère à son expérience, le RPC permet de récupérer un ensemble de plans cohérents et de les modifier par le biais de techniques d'adaptation évoluées pour arriver à un état désiré. Un plan est constitué d'un ensemble de préconditions et d'une séquence d'étapes déjà réalisées pour atteindre un but fixé. Le produit final du processus de planification comprend donc un ensemble d'étapes qui, une fois exécutées, sont censées produire l'état désiré du monde [Kolodner 93]. L'état désiré peut être décrit soit en termes concrets, ou en termes de contraintes qui doivent être satisfaites. L'utilisation du RPC pour résoudre ces problèmes de planification permet de restreindre l'espace de recherche pour élaborer de nouveaux plans. En effet, celui-ci est très vaste car il concerne des domaines d'application de la vie courante. Aussi, les problèmes étant souvent imprévisibles, les résultats sont moins concrets que ceux fournis par les tâches d'analyse et de fréquentes erreurs sont commises. Le RPC a la particularité de pouvoir conserver une trace des erreurs. Ainsi, un planificateur explique les erreurs et tente de représenter comment elles peuvent être réparées. Le système peut alors apprendre de ses erreurs. Le processus explicatif est très important dans ce type de système car il fait partie intégrante du raisonnement. En effet, pour pouvoir adapter des plans il est nécessaire au préalable de les comprendre, et donc de les expliquer. Ceci permet au système d'anticiper, et à chaque nouvelle étape il fait appel aux plans ayant déjà été confrontés à ce problème et peut ainsi anticiper les problèmes que ces plans-là ont rencontrés. Dans un problème de

classification classique, le succès de la recherche repose sur le choix des attributs sur lesquels va se faire la recherche. Il est nécessaire de bien indexer les cas. De plus le résultat consiste souvent en une seule caractéristique. Les tâches de planification ou de conception sont quant à elles étroitement liées à la bonne gestion des contraintes qui lient les attributs entre eux.
La tâche de conception consiste à construire un objet étant donné ses spécifications, souvent incomplètes. Ainsi, le système JULIA [Kolodner 93], exploite un répertoire de méthodes d'adaptation pour transformer les recettes antérieures pour rencontrer les contraintes du problème courant. Elle contient aussi des méthodes d'adaptation qui sont utilisées pour modifier et réparer les décisions passées invalidées par les contraintes. ARCHIE [Kolodner 93], un système RPC dans le domaine de l'architecture, possède des cas qui contiennent plusieurs types d'information incluant les buts de conception, les plans conçus, les descriptions de comment les modèles satisfont bien les buts et les contraintes et leçons devant être apprises du cas. Les problèmes de conceptions sont caractérisés par l'existence de contraintes. Celles-ci ne permettent pas de résoudre à proprement parlé le problème mais d'évaluer dynamiquement les résultats intermédiaires au fur et à mesure du processus de résolution. Les contraintes fournissent une ligne de conduite mais ne dirigent pas le raisonneur vers une solution particulière[Kolodner 93]. L'espace de recherche est tellement vaste qu'il faut savoir s'arrêter dès qu'une solution a été jugée satisfaisante. Il ne faut pas explorer tout l'espace de recherche car la quantité de solutions est immense. Pour les problèmes de conception aussi, la phase d'adaptation est très importante. Là aussi, les explications contribuent au succès de la résolution.
La tâche de configuration consiste, à partir d'un ensemble de composants donnés, à en sélectionner la totalité ou un sous-ensemble et de les assembler pour rencontrer un certain nombre de contraintes.

Nous venons donc de décrire les différentes tâches que le RPC permet de résoudre : les tâches d'analyse et de synthèse. Alors que les premières sont les plus courantes et les plus faciles à implémenter, les secondes se distinguent par la nécessité de prendre en compte la phase d'adaptation du RPC pour les réaliser. De plus, elles diffèrent par la complexité des cas qu'elles manipulent. Les cas utilisés pour la classification sont simples et précis, avec des attributs bien définis, tandis que ceux utilisés pour les tâches de synthèse contiennent des

données difficiles à représenter car souvent imprécises et incomplètes. De plus, nous avons vu que le rôle des explications dans les problèmes de synthèse est important.

Dans ce chapitre nous avons donc présenté les différents concepts sur lesquels s'appuie notre recherche. Tout d'abord, nous avons décrit les caractéristiques des SBC en général en montrant les points forts et les limites qui ont marqué le passage des SBC1 aux SBC2. De plus, nous avons présenté un type particulier de raisonnement, le RPC, les SBC-RPC représentant les systèmes auxquels nous nous intéressons. Nous allons maintenant, dans le chapitre suivant décrire la modélisation des explications dans les SBC-RPC ainsi que la méthodologie de développement de systèmes KADS que nous appliquerons pour nos travaux.

CHAPITRE III

Les explications dans les SBC-RPC

L'objectif de ce chapitre consiste dans un premier temps à décrire les différentes approches qui ont été suivies dans les travaux sur la génération d'explications dans les SBC-RPC. Ensuite, la méthodologie de développement de systèmes KADS présentant un intérêt pour notre sujet de modéliser les explications dans les SBC-RPC, une description de ses principes et des différentes phases de modélisation qu'elle comporte avec leur signification sera réalisée.

III.1 Approches actuelles de la production d'explications dans les SBC-RPC

Nous avons, dans le chapitre précédent, défini le RPC et ses caractéristiques. Celui-ci possède la particularité d'être plus facile à implémenter que les raisonnements classiques utilisés généralement dans les SBC, comme le raisonnement inductif par exemple [Watson 97; Kolodner 93]. Cependant, comme dans tout système, même si la méthode de résolution utilisée paraît simple et intuitive aux concepteurs du système, les utilisateurs peuvent être confrontés à certains problèmes de compréhension. C'est pourquoi il est apparu aussi dans le développement des SBC-RPC la nécessité d'apporter des capacités à fournir des explications pour pouvoir montrer aux utilisateurs, experts ou non, la pertinence des cas retrouvés. De même, les systèmes SBC-RPC développés étant davantage introduits dans des environnements de travail, expliquer le RPC est devenu une exigence importante [Goel *et al.* 96b]. Ainsi, la génération d'explications dans les SBC-RPC est, comme dans les SBC utilisant un autre type de raisonnement, liée au type des besoins, et s'avère donc diversifiée.

Il ressort des premiers résultats d'une étude bibliographique réalisée sur les travaux concernant la production d'explications dans les SBC-RPC, que les articles traitant exclusivement de ce sujet sont peu nombreux [Verrière 97]. En effet, les techniques et spécificités liées à la production d'explications dans de tels systèmes ne décrivent pas toujours les méthodes suivies, ni les systèmes implantés. On trouve peu de descriptions d'architecture de processus explicatif. Cependant, on peut constater, lorsqu'on feuillette la littérature portant uniquement sur le RPC, que les explications sont omniprésentes même si l'appellation explication n'est pas toujours mentionnée par les auteurs. En effet, souvent ceux-ci implantent des processus RPC dont les solutions au problème posé sont fournies avec les éléments pertinents ayant servi lors de la résolution. Étant donné que ces éléments apparaissent sous la forme de raisons, d'éléments descriptifs, de cheminement ou de raisonnement suivi pour arriver au résultat, on peut, dans une certaine mesure, considérer ces informations supplémentaires, fournies avec la solution, comme des explications. Une option d'aide est également souvent accessible, ce concept d'aide répond tout à fait à la définition du rôle d'une explication. De même, on peut considérer l'introduction de capacités d'argumentation [Aleven&Ashley 96] et de documentation technique [Kamp 93] dans des systèmes RPC comme une forme d'explication. Certes, les processus explicatifs ne sont pas toujours décrits explicitement et n'ont pas encore fait l'objet d'une implémentation, mais on peut remarquer qu'en général les explications sont prises en compte, au moins d'un point de vue conceptuel, lors de l'élaboration d'un système RPC, et ce dans divers domaines d'application. Ceci peut s'expliquer par le fait que le but des chercheurs en IA sur le RPC consiste à développer des théories pour construire des environnements à base de cas interactifs utiles et utilisables [Goel *et al.* 96a]. Dans ce qui suit nous distinguerons quatre axes de recherche dans les travaux sur les explications.

III.1.1 Les explications : un outil d'apprentissage

Dans des domaines mal structurés comme celui de la conception d'interfaces [Barber *et al.* 92] par exemple, la littérature existante est souvent assez pauvre. Aussi, l'utilisation d'exemples permet, d'une part d'expliquer les principes abstraits et les lignes de conduite à suivre qui sont généralement difficiles à comprendre et, d'autre part, d'illustrer leur utilisation, souvent rendue difficile lors de problèmes spécifiques. L'utilisation d'exemples constitue une approche utile pour structurer ce type de connaissances. Les cas déjà résolus représentant une grande source d'exemples, le RPC fournit une solution intéressante pour expliquer et

fournir des exemples de résolution de problème. Cette approche a été développée dans ASKJEF [Barber *et al.* 92], un prototype de système d'IA qui aide les ingénieurs de logiciels dans la conception d'interfaces Personne-Machine. De même dans [Weber 94], où il s'agit d'un système d'aide pour l'apprentissage d'un langage de programmation, les explications sont utilisées pour retrouver les problèmes de programmation déjà rencontrés et expliqués. Les explications sont alors utilisées comme outil de classification de problèmes et permettent à l'utilisateur, de par les informations qu'elles fournissent, d'y remédier. Les explications dans les travaux que l'on vient de citer sont utilisées comme outil, mais un outil de recherche qui permet d'aider l'utilisateur à comprendre sa tâche.

III.1.2 Les explications : un outil pour améliorer le raisonnement du système

La deuxième direction de recherche considère la production d'explications comme un outil destiné à améliorer le processus du RPC. Des techniques de développement de SBC-RPC à base d'explications sont alors utilisées pour justifier les actions d'un cas avec respect pour les caractéristiques connues quand les cas ont été originellement exécutés. [López de Mántaras&Plaza 97]. Les explications ne sont donc plus produites dans l'optique d'aider l'utilisateur final mais leur rôle consiste principalement à améliorer ou à évaluer le processus RETROUVER du RPC. En effet, nous avons énoncé que le RPC peut être utilisé pour des problèmes de classification, pour des domaines où les données sont bien connues, mais aussi pour des problèmes de synthèse, dans des domaines plus complexes où la connaissance est générale, donc pas entièrement disponible. C'est dans ces domaines, où l'on traite des tâches de planification, de conception, que les techniques pour retrouver les cas, les adapter ou les réparer posent le plus de difficultés. Plus particulièrement, on retrouve ce type d'explications dans des domaines où les systèmes développés sont des systèmes de recherche basés sur la connaissance. Ils sont le résultat de la combinaison des techniques du plus proche voisin et des techniques basées sur la connaissance et ils sont caractérisés par l'utilisation de la connaissance pour construire des explications montrant pourquoi un problème a eu une solution particulière dans le passé [Bento *et al.* 94a, 94b; Macedo *et al.* 96]. Dans ces systèmes, les explications sont contenues dans les cas et permettent, comme nous l'avons mentionné plus tôt, d'améliorer et de guider le processus de recherche. C'est pourquoi, l'utilisation d'explications de situations antérieures, pour retrouver des cas, expliquer les similarités, évaluer la pertinence, trouver les erreurs et/ou les réparer, fait l'objet de nombreux travaux. En général un cas est représenté sous forme d'un triplet contenant le problème décrit comme

un ensemble de faits, la solution en un ou plusieurs faits selon le cas et une justification des liens entre les deux. Celle-ci peut correspondre à une chaîne d'inférence qui lie ensemble plusieurs parties d'un cas [Kolodner 93] ou à un ensemble d'explications qui sont représentées sous forme d'un arbre de preuve liant les prémisses (ou données du problème) aux faits de la solution [Bento *et al.* 94a,94b]. Dans l'approche suivie par Bento, la théorie du domaine n'étant pas complète, les explications des cas survenus sont considérées comme pouvant être imparfaites. Ainsi, il utilise la notion d'empreinte pour désigner le sous-ensemble des faits d'un cas qui sont associés à un fait de la solution par une explication. La qualité de cette explication, à savoir si elle est complète, incomplète ou partielle, donne le degré de certitude à l'empreinte associée. Ainsi, lorsqu'un cas est recherché dans la base de cas, une fonction d'évaluation calculant la différence entre les ressemblances et les dissemblances des empreintes contenues dans chaque cas en fonction de leur degré de certitude est calculée. Les cas retrouvés sont expliqués en utilisant une comparaison des empreintes contenues dans chacun d'eux. Le système qui utilise cette méthode est RECIDE [Bento *et al.* 94a]. De la même manière, [Muñoz-Avila&Hüllen 96] présente un algorithme pour expliquer la performance des cas retrouvés. Il utilise aussi la notion d'empreinte. Il introduit un modèle de pondération des caractéristiques. Les critères pertinents sont indexés de telle manière que ceux qui ont le meilleur degré de pertinence sont assortis en premier. Une caractéristique est considérée comme pertinente à un but par rapport à une solution si la caractéristique contribue à atteindre le but dans la solution. Dans les travaux de [Reategui *et al.* 95,97], qui réalise des systèmes hybrides, réseaux neuronaux et RPC, un principe similaire est développé en représentant la connaissance générale sous forme de descripteurs de diagnostics qui sont gardés et utilisés par le système pour déterminer si une réponse finale est crédible et pour construire des explications du raisonnement. Les descripteurs de diagnostic doivent représenter et mettre en valeur les caractéristiques les plus importantes pour l'identification d'un diagnostic, et pour associer le diagnostic à des expériences passées. Ainsi, on peut constater qu'en se basant sur la qualité des explications retrouvées, on peut juger de la similarité entre les cas. L'approche développée dans [Macedo *et al.* 96] considère les liens entre les cas. Chaque nouveau cas est expliqué par des liens antécédents et explique des morceaux d'autres cas par les liens conséquents. Le domaine d'application étant la planification, un cas est un plan sous forme d'un ensemble structuré de buts et d'actions. Les morceaux de cas sont reliés entre eux par des liens hiérarchiques et temporels (explication)

qui forment un réseau (arbre). Le fait de fractionner les cas en morceaux de cas qui sont expliqués permet d'améliorer les performances du système. Les travaux de [Ashley&Aleven 93] ont aussi pour objectif d'expliquer les facteurs de pertinence des cas retrouvés. La méthode consiste à expliquer les similarités et les différences des critères pertinents représentés dans la logique du premier ordre sous la forme de facteurs. Les cas pertinents sont illustrés sous forme d'exemples. Toutes ces approches se caractérisent par le fait que les explications existent dans les cas et ce dès la conception. Comme nous l'avons mentionné, elles représentent des techniques basées sur les explications pour améliorer le processus retrouver du RPC. Cependant, si elles permettent d'expliquer les similarités entre les cas retrouvés, elles ne permettent pas d'apprendre de leurs erreurs.

III.1.3 Les explications : un outil permettant au système d'apprendre de ses erreurs

Nous situant dans des domaines complexes où la connaissance est générale, les problèmes sont alors décrits de manière plus ou moins explicites, c'est pourquoi les cas retrouvés ou même les solutions trouvées peuvent être erronées. Il faut alors réparer les erreurs. Lorsqu'on arrive à trouver une erreur, cela signifie qu'on l'a comprise, et qu'elle peut donc être expliquée. En intégrant cette explication dans le cas, une consultation future de celui-ci rendra compte de ces éléments et permettra alors que la même erreur ne soit pas répétée. Du fait que, contrairement à d'autres types de raisonnement, le RPC enrichisse au fur et à mesure des résolutions sa base de cas, il acquiert naturellement des capacités d'apprentissage, ce qui lui permet de conserver un historique de cette base. En effet, les bases de cas sont dynamiques, comme les BD, de nouveaux enregistrements sont conservés d'une résolution à l'autre. Aussi, lorsqu'un cas est résolu, la justification de la solution - à savoir sur quels critères de similarité il a été obtenu, à partir de quels cas retrouvés, selon quelles données du problème - peut être construite et insérée directement dans le cas. Ce processus, à lui seul, fournit un dispositif pour générer des explications. En effet, déjà par nature, un cas contenant les données du problème et la solution associée représente à lui seul une explication. Si on lui ajoute de l'information sur la manière dont il a été construit, on enrichit l'explication. L'approche hybride développée par Agre [Agre 95] combine un système à base de règles (SBR) et le RPC. Le RPC est utilisé pour corriger les erreurs réalisées par le SBR en retrouvant les erreurs similaires déjà commises dans le passé. Les erreurs sont réparées en utilisant un vocabulaire d'erreurs sous forme de prédicteurs d'erreurs. Ceux-ci permettent d'expliquer ces dernières, et permettent d'éviter de répéter les mêmes anomalies dans le futur. L'approche

développée par Cox [Cox&Ram 94] consiste à considérer l'apprentissage comme une tâche de planification. Le domaine considéré est la compréhension d'histoires. Il utilise des patrons d'explications pour expliquer les anomalies détectées. D'après Schank [Schank *et al.* 94], un patron d'explication est, par essence, une explication banale qui a été utilisée auparavant, et qui peut être adaptée durant le processus d'explication pour être utilisée dans de nouvelles circonstances. Ce processus d'adaptation d'explications standards pour une utilisation dans de nouveaux espaces permet la créativité [Schank *et al.* 94]. Aussi, face à une anomalie, des patrons d'explications qui correspondent au schéma sont retrouvés, permettant ainsi d'identifier le problème. En justifiant les erreurs par une explication, et en modifiant les patrons, le système acquiert d'autres connaissances. Le système développé, META-AQUA, détecte, explique, répare et apprend de son raisonnement sur les erreurs. Les travaux entrepris par Leake [Leake 95] vont plus loin, l'objectif consistant à ce que le système puisse aussi, à terme, raisonner sur son propre raisonnement. *Nous ne voulons pas uniquement expliquer ce que nous ne comprenons pas, nous voulons pouvoir faire une généralisation des événements expliquant le phénomène précis afin de pouvoir réutiliser cette connaissance ultérieurement* [Schank *et al.* 94]. Ainsi, l'approche développée par Schank consiste à expliquer les anomalies de la vie courante. Pour cela, il utilise une approche basée sur l'abduction et sur la généralisation. On retrouve cette approche dans le système SWALE [Leake 95]. Le principe est le suivant, à partir de la description d'une anomalie de la vie courante et du but à atteindre, un ensemble de patrons d'explications contenant les critères associés à la situation en cours est recherché. À partir des patrons retrouvés, des hypothèses sont émises pour trouver la solution au problème courant conformément aux explications contenues dans les patrons, c'est la phase d'adaptation. Elles sont ensuite vérifiées, et, si elles conviennent, le nouveau cas solutionné est intégré dans la base de cas. L'apprentissage et la résolution de problème reposent donc sur le cycle du RPC basé sur les explications. On retrouve également ce principe dans la méthode développée par Aamodt [Aamodt 93] qui utilise la connaissance générale du domaine comme support pour générer des explications pour la résolution de problème à base de cas et l'apprentissage. Une méthode de raisonnement générique, introduite dans un cycle activer-expliquer-cibler, est capable d'utiliser un modèle de connaissance riche pour produire des explications dépendantes du contexte. Chaque tâche du RPC est réalisée selon ce cycle et les cas sont combinés à la connaissance générale du domaine. La particularité des problèmes qui sont traités est qu'ils sont définis par un but qui

se décompose en une ou plusieurs tâches. Cette décomposition en tâches du processus du RPC est reprise dans les travaux de Grimnes [Grimnes&Aamodt 96].

III.1.4 Les explications : un outil permettant à l'utilisateur de comprendre la tâche du système

Le principe de la modélisation comme support pour expliquer le raisonnement et les connaissances constitue la dernière orientation que nous citerons. Elle a été développée dans [Goel *et al.* 96a, 96b, 97] où l'hypothèse suivie part du principe que l'utilisateur, pour utiliser plus facilement les SBC-RPC, doit pouvoir former un modèle mental de la manière dont le système travaille. Ce dernier doit alors être capable d'expliquer son raisonnement et de justifier ses réponses. Pour cela, une méthode à base de modèles est développée. Deux modèles sont utilisés. Le premier, le modèle tâche-méthode-connaissance permet d'obtenir des explications aux niveaux de la tâche et des connaissances. Il permet de capturer le contenu fonctionnel et stratégique du raisonnement ainsi que le contenu de sa connaissance. Le second, le modèle structure-comportement-fonction, permet d'expliquer les fonctionnalités, les causalités de même que la structure d'un système. Il fournit des explications à un niveau d'abstraction permettant une communication efficace entre le système et l'utilisateur. La dernière orientation prise dans le développement de systèmes explicatifs consiste donc notamment à fournir des explications du raisonnement suivi par le système, soit du RPC. Ce principe de modélisation des tâches est propre aux méthodes de conception des SBC telles que la méthodologie CommonKADS [Schreiber *et al.* 94] qui est utilisée dans les travaux de Grimnes ou KADS.

Ainsi, nous avons pu constater que la génération d'explications dans les SBC-RPC faisait l'objet de travaux très diversifiés. Elles peuvent intervenir dans un contexte de support d'aide, comme outil pour améliorer les performances des processus du RPC, en tant qu'outil d'apprentissage permettant au système de réparer et d'apprendre de ses erreurs ou enfin, et c'est le cadre dans lequel nous nous situerons dans notre travail, comme outil d'apprentissage pour l'utilisateur et pour expliquer à l'utilisateur les connaissances manipulées et le raisonnement utilisé. L'utilisation d'une méthodologie de développement étant, comme nous venons de le décrire, appropriée pour traiter ce problème, nous allons dans le paragraphe suivant décrire la méthodologie KADS.

III.2 La méthode KADS

KADS est une méthode d'ingénierie des connaissances basée sur la modélisation des connaissances, contrairement au prototypage rapide utilisé dans les premières méthodes de développement de SBC basées sur l'implémentation des connaissances [Dieng 93]. KADS a été élaborée dans le cadre du projet européen ESPRIT-I [Schreiber *et al.* 93], en particulier par des chercheurs d'Amsterdam [Wielinga *et al.* 92]. Construite à partir de la synthèse et de l'amélioration de nombreuses techniques en utilisant le paradigme de modélisation explicite des connaissances, KADS représente aujourd'hui un standard pour le développement de SBC. D'ailleurs d'autres méthodes ont été développées à partir de KADS. KADS-II aussi appelée CommonKADS [Schreiber *et al.* 94] *est* le résultat du projet ESPRIT-II, suite du projet ESPRIT-I, et vise à améliorer la première version de KADS (KADS-I) [tansley&Hayball 93] notamment en couvrant le cycle de vie complet du développement des SBC. Nous allons dans la section suivante décrire les principales caractéristiques et phases de développement de la méthode KADS, ensuite nous considérerons la place des explications au sein d'une telle méthode de développement de logiciel.

III.2.1 Les phases de développement

Dans KADS, le développement d'un SBC est vu comme l'élaboration d'un ensemble de modèles structurés. Une analyse préalable de l'application est réalisée avant d'entreprendre les étapes de conception et d'implémentation. Le principal handicap dans l'élaboration des SBC est lié, comme nous l'avons mentionné dans le chapitre II, à la difficulté à recueillir et à interpréter la connaissance des experts sous une forme qui permette aux ingénieurs de la connaissance d'avoir une meilleure compréhension du domaine afin qu'ils puissent la traduire de manière utilisable pour un système informatique. En séparant le niveau des connaissances du niveau des symboles [Newell 82], par le biais de l'élaboration de modèles structurés, cette difficulté est diminuée et une meilleure réutilisabilité des connaissances est possible. De plus, la flexibilité, la maintenance et la robustesse des SBC sont améliorées [David 95]. En effet, les connaissances sont acquises selon différents niveaux d'abstraction permettant ainsi une représentation plus explicite et plus adaptée selon les différentes tâches que le SBC va réaliser, ou encore pour satisfaire la fonction majeure de l'application. Cette notion d'abstraction a également été reconnue comme importante pour l'élaboration de dispositifs explicatifs [Bourcier *et al.* 94].

Le développement de la méthodologie n'a cessé d'évoluer depuis le début de son existence. Aussi, si KADS-I traitait essentiellement des phases d'analyse et de conception du traditionnel cycle de vie d'un logiciel [Tansley&Hayball 93], la version KADS-II, aussi appelée CommonKADS [Schreiber *et al.* 94], couvre l'essentiel de ce dernier. En effet, les phases d'implémentation, d'installation, d'utilisation, de maintenance et de raffinement des connaissances sont également prises en compte dans CommonKADS. L'effort entrepris dans CommonKADS réside donc dans le développement de la méthode pour la totalité du cycle de vie, mais aussi dans une meilleure explicitation des modèles générés afin que la méthode soit plus accessible et donc plus facilement applicable. Ainsi, l'évolution de l'approche se caractérise par le fait qu'elle inclut maintenant des dispositifs approfondis pour modéliser la connaissance du domaine, ce qui permet également d'accéder à une connaissance plus explicite en matière d'explications. Finalement, ce qui caractérise la méthodologie KADS en général, outre la modélisation, est qu'elle fournit un support additionnel sous forme de méthodes et de techniques pour tenir compte d'activités comme la modélisation de l'expertise ou l'extraction des connaissances, l'analyse de l'expertise étant une phase non supportée dans le cycle de vie traditionnel de développement de logiciels. Cette phase d'analyse constitue la première phase de modélisation de la méthodologie KADS. Elle consiste à élaborer un ensemble de modèles qui une fois terminés seront utilisés directement lors de la phase de conception. L'ensemble des divers modèles produits au cours de ces deux phases, qui vont être détaillées dans les deux paragraphes suivants, est illustré par la figure 4.

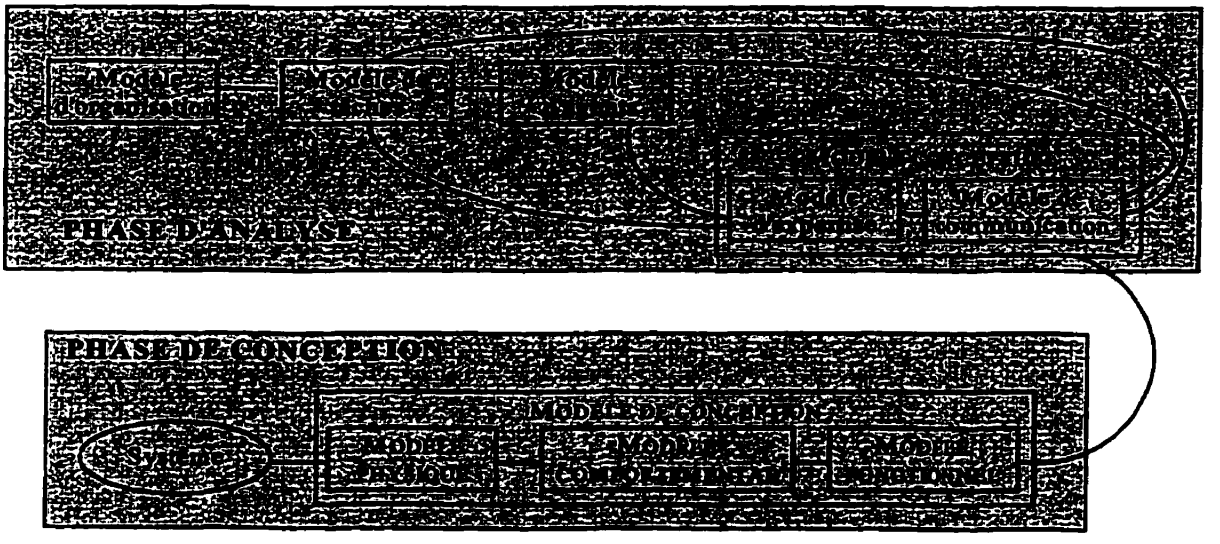

Figure 4. Les modèles de KADS

III.2.1.1 La phase d'analyse

La phase d'analyse est orientée vers le monde plutôt que vers le SBC. Elle s'effectue selon trois flux [Tansley&Hayball 93; Simian&Tourigny 96]. Tout d'abord l'analyse externe permet d'élaborer le modèle de l'organisation ainsi que le modèle des tâches. Le premier contient le modèle des besoins de KADS-I, mais permet également de définir les caractéristiques globales de l'organisation. À son issue, on connaît le problème, la solution ainsi que les contraintes d'environnement. Ensuite, l'analyse des modalités permet de définir les modèles de communication et d'agent, car il définit les flux entre agents ainsi que les distributions de tâches parmi eux et les contraintes au sein de l'organisation. CommonKADS se distingue effectivement de KADS-I par le fait que la première tient compte de manière très spécifique de l'environnement organisationnel d'une application. En effet, en général, la production d'un SBC ne permet de réaliser qu'une fraction des tâches existantes au sein d'une organisation. De plus, le développement de ces tâches prend généralement davantage en compte les aspects techniques que les facteurs sociaux ou organisationnels pourtant décisifs pour la réussite des résultats [Schreiber *et al.* 94]. C'est pourquoi, afin de limiter les erreurs et de garder une trace globale de l'environnement dans lequel le système doit opérer, CommonKADS permet de représenter l'environnement organisationnel. Enfin, l'analyse interne permet de procéder à l'acquisition des connaissances portant sur la résolution experte du problème. Elle consiste à analyser les connaissances du domaine en terme de concepts et de hiérarchies de concepts (connaissances statiques) ainsi que les tâches de résolution de problème par le biais de la sélection d'un modèle d'interprétation, et a pour résultat le modèle d'expertise, un modèle à quatre niveaux de connaissances qui permet de spécifier l'expertise de résolution de problème nécessaire à l'exécution des tâches. Une des caractéristiques de la méthodologie réside dans l'utilisation d'une bibliothèque de tâches génériques. Ainsi, avant de commencer l'analyse interne, un modèle de la tâche, appelé modèle d'interprétation, est choisi dans cette bibliothèque en fonction de la tâche à réaliser et est ensuite instancié au cours de l'analyse. Ces modèles sont généralement constitués de données sur les quatre niveaux du modèle d'expertise. L'utilisation de ces modèles représente le point fort de la méthodologie car ils permettent une réutilisation et une standardisation des modèles de tâches [Tansley&Hayball 93]. Le résultat de la phase d'analyse est donc constitué de cinq modèles : le modèle de l'organisation, le modèle des tâches, le modèle d'agents, le modèle d'expertise et le modèle de communication, ces deux derniers formant le modèle conceptuel. La structure générale d'un

modèle est définie dans la figure 5. Ainsi, la construction de chacun des modèles est réalisée en prenant en compte les caractéristiques définies dans le patron général d'un modèle. De plus, les modèles sont reliés entre eux par des relations, car certaines de leurs caractéristiques sont directement dépendantes ou dérivées les unes des autres. La structure globale des dépendances entre modèles est représentée dans la figure 6.

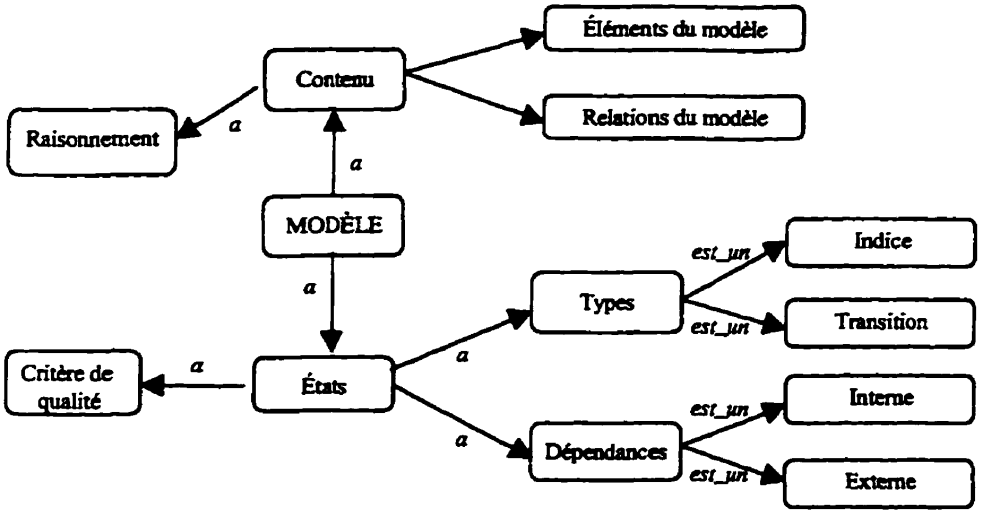

Figure 5. Structure générale d'un modèle CommonKADS [DeHoog *et al.* 94]

Nous allons décrire dans ce qui suit chacun de ces modèles.

➢ ***Le modèle de l'organisation*** :

Dans la méthode CommonKADS, on part du principe que le développement de SBC entraîne la construction d'un ensemble de modèles qui traitent non seulement de la connaissance de l'expert du domaine, mais aussi des diverses caractéristiques concernant la manière dont cette connaissance est impliquée et utilisée dans son environnement organisationnel [Schreiber *et al.* 94]. Ce modèle fournit les aspects principaux de l'environnement organisationnel en représentant une connaissance à plusieurs niveaux de cette organisation. Elle consiste à décrire l'organisation en respectant la structure du modèle décrite en figure 5 à l'aide de composants spécifiques et de relations entre eux. L'un des schémas typiques de construction du modèle est le suivant [De Hoog *et al.* 94]. Tout d'abord, les caractéristiques générales de l'organisation sont identifiées, c'est la phase d'introduction à l'organisation. Ensuite, la compréhension générale de la structure et du processus de l'organisation est acquise. Ceci permet de décrire ce qu'on appelle les constituants variants d'une organisation, soit les fonctions existantes, la structure de l'organisation, mais aussi d'identifier les agents évoluant, le processus selon lequel l'organisation fonctionne et les ressources disponibles. Ensuite, une

analyse détaillée permet de recueillir suffisamment de données pertinentes pour entamer une étude de faisabilité détaillée. Cette étape permet de compléter l'étude des éléments précédemment définis et d'identifier les problèmes et les opportunités de l'organisation. Enfin, l'évaluation de l'analyse réunit tous les éléments définis et permet de choisir des solutions au problème à résoudre.

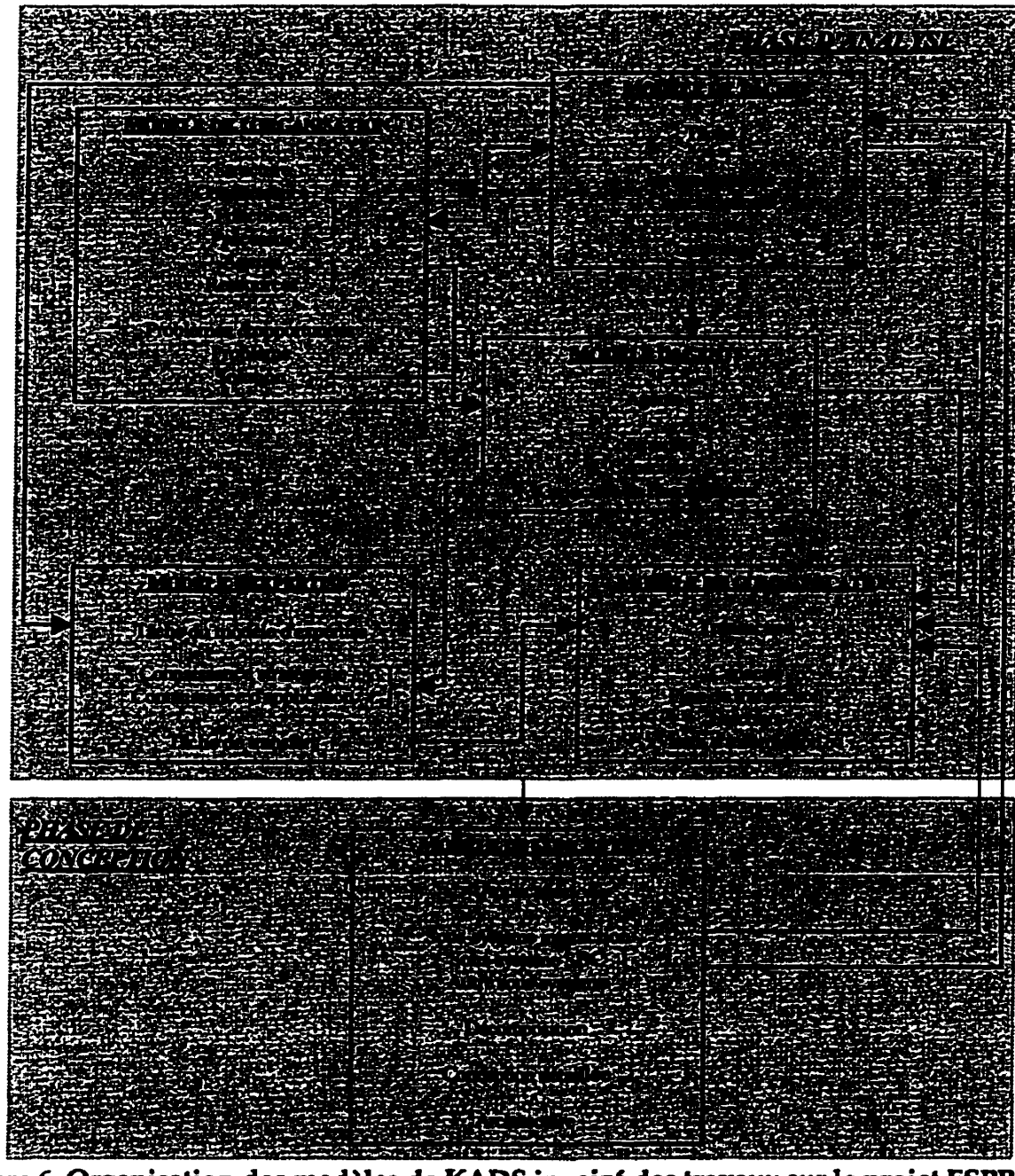

Figure 6. Organisation des modèles de KADS inspiré des travaux sur le projet ESPRIT P5248 KADS-II [De Hoog *et al.* 94; Duursma *et al.* 94; Van de Velde 94; Wærn *et al.* 93; Wærn&Gala 93; Wielinga 94]

Une autre stratégie consiste à commencer par la modélisation de la situation future. Elle consiste tout d'abord à identifier une liste de solutions cohérentes de SBC à partir de la littérature ou d'expériences similaires antérieures. À la fin de cette étape, les problèmes et opportunités ainsi que les solutions sont identifiées. Ensuite, l'étude de faisabilité permet de décrire la nouvelle situation en identifiant les ressources informatiques et la fonction de l'organisation. Enfin, une prise de décision permet de définir les priorités et ainsi de valider les solutions avant de décrire le problème.

➢ ***Le modèle de tâches***

Le modèle de tâches est le résultat d'un processus d'analyse de tâches qui consiste à identifier dans l'environnement organisationnel une décomposition hiérarchique des tâches à réaliser pour résoudre le problème posé en fonction des activités et des capacités des agents [Duursma *et al.* 94]. L'analyse des tâches se fait en trois étapes [Duursma *et al.* 94]. Tout d'abord, il faut faire une description du problème général en fonction des éléments organisationnels recueillis. Une fois la tâche à réaliser identifiée, il faut procéder à une décomposition hiérarchique de la tâche en sous-tâches en décrivant pour chacune : son type, les informations en entrée et en sortie, le but à atteindre, les sous-tâches et la tâche supérieure auxquelles elle est reliée. Ceci constitue le modèle de tâches initial. La deuxième étape consiste à étudier les besoins ou conditions associées à la réalisation de chacune des tâches. Elle fait appel au modèle de communication. Ainsi on détermine d'une part, la distribution des tâches aux agents d'après leur fonction dans l'organisation, de leurs activités et de leurs capacités et d'autre part, les relations entre tâches dans l'organisation générale. Le résultat constitue le modèle des tâches nécessaires. Enfin, la dernière étape consiste à améliorer le modèle des tâches proposé en vérifiant que le but donné de la tâche est réalisable conformément aux activités de la tâche. Si ce n'est pas le cas, il faut réviser le modèle soit en redéfinissant le but à atteindre, soit en redécomposant la tâche en sous-tâches qui lui permettent d'atteindre le but tel qu'il est énoncé. Le modèle issu de cette phase est le modèle courant. Tout changement apporté au modèle de tâches doit être associé à une mise-à-jour des modèles avec lesquels il est en corrélation tels que le modèle d'agent, de communication ou d'expertise. En effet, la figure 6 montre que les modèles sont liés les uns aux autres par des relations. Pour élaborer le modèle de tâches on n'utilise pas de modèle formel, mais un modèle générique dont les caractéristiques sont instanciées lors de l'analyse d'une tâche spécifique en fonction de leur importance pour l'exécution de celle-ci. L'analyse de tâches

représente un outil pour élaborer la documentation destinée à l'utilisateur. Finalement, l'analyse de tâches a pour rôle d'aider l'ingénieur de la connaissance à organiser une vue d'ensemble des tâches principales de l'expert dans un domaine donné, d'aider à définir le domaine de la solution du SBC et à supporter l'analyse de faisabilité du projet. L'analyse des tâches n'est pas spécifique à CommonKADS, elle est réalisée dans toutes les méthodes de développement.

➢ ***Le modèle d'agents***

Ce modèle consiste à allouer à des agents spécifiques les tâches définies dans le modèle de tâches [Wærn&Gala 93]. Un agent qui peut être représenté entre autres par le système ou l'utilisateur a pour rôle d'exécuter des tâches. Ce modèle sert en fait de lien entre les modèles de tâches, de communication et d'expertise en modélisant les capacités de raisonnement, les contraintes, sous forme de normes, de préférences ou de permissions, ou encore toutes les propriétés pertinentes des agents qui sont impliqués dans la résolution d'une tâche. La construction du modèle d'agent se fait de la façon suivante [Wærn&Gala 93]. 1) Tout d'abord il faut établir l'ensemble des agents. On les distingue selon trois catégories : a) ceux qui vont être développés au cours du projet, b) ceux qui existaient déjà au sein même de l'organisation, c) les utilisateurs. 2) Ensuite, il faut effectuer une classification des agents. Ainsi, on peut modéliser les agents définis dans le modèle d'organisation en fonction du rôle et des activités qu'ils réalisent dans l'organisation. Par exemple, dans le domaine médical on ne classifiera pas dans le même modèle les docteurs et les patients. On peut aussi classifier les agents en fonction des tâches spécifiques à exécuter. 3) L'étape suivante consiste à modéliser les agents utilisateurs. Cette étape doit avoir lieu le plus tôt possible, car les informations recueillies peuvent influencer le développement du système global. En effet, cette phase est d'autant plus importante que les utilisateurs du système final ne sont pas forcément les experts du domaine qui ont fourni la connaissance et toute l'expertise à l'ingénieur de la connaissance. Ils n'ont donc pas le même profil. 4) Finalement il faut modéliser les agents de type système, mais ceci est généralement fait lors de l'élaboration des modèles de communication ou d'expertise du fait que cela nécessite des connaissances sur la modélisation conceptuelle représentée dans le modèle conceptuel. Le modèle d'agents correspond à une partie du modèle de coopération du modèle KADS-I. Dans CommonKADS il représente un moyen de traiter explicitement les problèmes liés aux caractéristiques du système ou des utilisateurs. Ce modèle permet donc d'expliciter clairement le modèle de

l'utilisateur impliqué dans la résolution des tâches. Le modèle de l'utilisateur contenant la connaissance générale du domaine des utilisateurs, leurs préférences, leur niveau de compétences ou autre fait partie des caractéristiques utiles à la génération d'explications.

➢ ***Le modèle de communication***

La construction du modèle de communication a pour rôle de spécifier l'environnement communicatif dans lequel le SBC doit fonctionner. Il décrit les échanges d'information entre les agents qui exécutent la même tâche, en fournissant les concepts et les mécanismes de communication en fonction des conditions établies et des compétences des agents communicants. La construction de ce modèle permet d'évaluer, d'une part, les risques associés au bon fonctionnement du système, et de valider, d'autre part, que les interactions personne-machine sont d'une qualité acceptable pour éviter ces risques. Le processus de construction ne peut être entrepris que si une première version du modèle de tâches contenant l'assignation de celles-ci, en accord avec les spécifications du modèle d'agent et l'identification des ingrédients à transférer entre agents, est terminée [Wærn *et al.* 93]. La construction du modèle s'effectue en cinq phases. La première consiste à identifier et à décrire, d'une part, l'ensemble des transactions qui représentent le transfert d'ingrédients ou items d'information entre leur propriétaire et leur(s) destinataire(s) et, d'autre part, les relations qui relient les transactions aux constituants d'autres modèles. La deuxième phase a pour rôle de décrire les plans de transaction qui détaillent les conditions et les contraintes sur les transactions et leur ordre d'exécution. Ensuite, la troisième phase ou description des constituants du discours pour les transactions permet de définir un plan décrivant comment effectuer une transaction comme un ensemble d'interactions uniques ou d'actes du langage. C'est au cours de cette phase qu'est déterminé le type d'interface de communication qui sera utilisé (questions/réponses, menus, etc.). La phase suivante consiste à identifier et à décrire les items d'information qui définissent comment chaque transaction devrait être exécutée ainsi que la syntaxe des différents actes du discours qui interviennent dans le discours. Finalement, la dernière phase consiste à décrire les capacités que doivent posséder les agents pour réaliser les transactions en fonction de la structure du discours et du langage d'information utilisé. Le modèle de communication sert donc de spécification pour la modélisation de l'interface. En effet, son rôle consistant également à répondre à des questions relatives aux questions personne-machine, il doit donc fournir un moyen pour le traitement des explications, de l'initiative, du dialogue, des médias et des groupes d'utilisateurs. De

plus, il permet de vérifier la cohérence des informations véhiculées entre les systèmes de tâches, d'agent et d'expertise et est souvent à l'origine des modifications apportées dans ces derniers. Finalement, le modèle de communication de CommonKADS correspond essentiellement au troisième niveau du modèle de coopération de KADS-I qui est le niveau de la communication.

➢ ***Le modèle d 'expertise***

La construction du modèle d'expertise représente l'activité principale de la méthode KADS. Elle permet de distinguer le développement des SBC de celui des systèmes traditionnels dans lesquels cette phase n'existe pas. Son but consiste à spécifier l'expertise de résolution de problème nécessaire à l'exécution des tâches de résolution de problèmes assignées au système [Schreiber *et al.* 93]. Le modèle d'expertise est vu comme un modèle exprimé au niveau des connaissances, c'est-à-dire un modèle qui rend l'organisation de la connaissance dans le système explicite à travers une typologie des connaissances élaborée [Schreiber *et al.* 94]. Le rôle d'un tel modèle est de pouvoir expliquer, dans un vocabulaire qui soit compréhensible par les utilisateurs, comment et pourquoi le système va réaliser une tâche. L'intérêt porte donc sur le comportement de résolution de problème du système et sur les types de connaissances impliquées dans la génération d'un tel comportement, abstraction faite des détails d'implémentation [Wielinga 94]. Ainsi, le modèle d'expertise repose sur la spécification des méthodes de résolution de problèmes et sur la catégorisation à un haut niveau d'abstraction des connaissances impliquées. Cette catégorisation de la connaissance permet de distinguer la connaissance du domaine ou la connaissance statique représentée dans le premier niveau du modèle le niveau domaine, de la connaissance de contrôle correspondant aux niveaux inférences et tâches du modèle. Ainsi, le modèle d'expertise est structuré en quatre niveaux de connaissance : domaine, inférence, tâche et stratégie. La construction du modèle d'expertise est réalisée pendant l'étape d'analyse interne du système selon trois phases : tout d'abord, la connaissance statique est analysée, ensuite ce sont les tâches de résolution de problèmes pour guider la sélection d'un modèle d'interprétation et enfin le modèle d'expertise à quatre niveaux est construit [Simian&Tourigny 96; Schreiber *et al.* 93, 94]. Ceux-ci sont décrits ci-après.

➢ *Le niveau du domaine*

Ce niveau contient la théorie du domaine c'est-à-dire la connaissance décrite par les concepts, les relations, les propriétés et les structures du domaine, et c'est ce qu'on appelle la

connaissance statique. La méthode CommonKADS se distingue notamment de KADS-I par l'introduction d'ontologies. Ainsi, dans CommonKADS, le modèle d'expertise supporte l'introduction d'ontologies comme un mécanisme pour générer des descriptions abstraites de la structure et du vocabulaire de la connaissance du domaine. Les ontologies relient aussi la connaissance du domaine avec la connaissance d'inférence, expliquant et minimisant ainsi les interactions entre la connaissance déclarative et la tâche.

- *Le niveau des inférences*

Il contient avec le niveau de la tâche la connaissance de contrôle. Ce niveau décrit comment utiliser la connaissance du domaine selon des étapes de raisonnement élémentaires ou inférences. C'est pourquoi la connaissance d'inférence est modélisée en termes d'opérations sur la connaissance du domaine et en terme de rôles. Dans KADS-I les rôles qui sont joués par un ensemble de connaissance dans un processus d'inférence sont décrits par les métaclasses, et les étapes d'inférence au niveau des inférences sont appelées sources de connaissances. Ainsi, une source de connaissance caractérise la transformation d'une métaclasse en une nouvelle métaclasse [Schreiber *et al.* 93; Simian&Tourigny 96]. Les inférences sont représentées par leur nom, une spécification d'entrée/sortie et une référence à la connaissance du domaine utilisée.

- *Le niveau de la tâche*

Les connaissances de ce niveau, appartenant aussi aux connaissances de contrôle, décrivent comment décomposer la tâche de raisonnement de haut niveau et comment imposer le contrôle sur cette décomposition pour atteindre un but particulier. Elle indique comment exécuter et séquencer les inférences pour atteindre un objectif et donc comment appliquer la connaissance des deux plus bas niveaux dans les activités de résolution de problème du domaine. Dans CommonKADS, ces connaissances sont représentées selon une hiérarchie de tâches. Les feuilles de cette hiérarchie invoquent des inférences particulières. La spécification d'une tâche se décompose en deux parties. La première est relative à la définition de la tâche, elle spécifie de manière déclarative le but à atteindre. La deuxième traite du corps de la tâche, elle spécifie les procédures et les activités à réaliser pour accomplir la tâche. Dans CommonKADS, la spécification du corps d'une tâche peut être réalisée par le biais des méthodes de résolution de problèmes. En effet, CommonKADS contient une librairie de méthodes de résolution de problème. Celles-ci permettent de ne pas commencer l'analyse de l'expertise depuis le début mais d'utiliser des méthodes de résolution de problèmes déjà

existantes si la définition de la tâche à décomposer correspond aux critères d'acceptation et de capacité de la méthode. Ces méthodes décomposent une tâche en sous-tâches ou fournissent une méthode pour atteindre directement une solution.

➢ *Le niveau stratégique*

La connaissance du niveau stratégique contient les connaissances relatives à la manière de sélectionner les tâches appropriées pour la résolution de problème pendant l'exécution du plan. Elle détermine les buts pertinents à la résolution d'un problème particulier. Elle porte sur la connexion des différentes tâches, la gestion des erreurs des méthodes de résolution utilisées, etc.

➢ ***Le modèle conceptuel***

Le modèle conceptuel représente le modèle de résolution de problème adapté au problème du système expert. Il représente la donnée d'entrée du processus de conception. Il est constitué du modèle d'expertise et du modèle de communication qui représentent à eux deux la base conceptuelle pour la conception technique du système [Schreiber *et al.* 94]. En fait le modèle conceptuel est une traduction de problème réel sous une forme abstraite. Dans KADS-I, ce modèle est constitué du modèle d'expertise et du modèle de coopération. Le modèle de coopération de la méthode KADS-I est en fait la réunion des modèles de l'agent, de tâches et de communication. Il intègre aussi une caractéristique établissant qui doit prendre l'initiative des transferts entre les agents. En effet, le modèle de coopération est un modèle à trois niveaux, le niveau de la tâche qui correspond au modèle des tâches de KADS-II, le niveau sémantique qui permet de définir les relations entre le modèle d'agent et le modèle de tâches, il concerne donc le modèle d'agent, et enfin un niveau de communication qui est le modèle de communication de KADS-II [Wærn *et al.* 93], et ce même s'il est souvent défini comme ne représentant que les échanges entre agents. Finalement, une fois que la phase de conceptualisation est terminée, il ne devrait pas y avoir de retour sur celle-ci, cependant, le raffinement des modèles exige l'alternance de recueils et d'analyse de ceux-ci [Martin 94].

Le produit final de la phase d'analyse, appelé modèle d'analyse, est donc constitué de l'ensemble des modèles que nous venons de définir, à savoir les modèles de l'organisation, de tâches, d'agent, de communication et d'expertise. Il offre un support de base pour commencer

la phase de conception. Dans le chapitre VI, nous verrons les liens entre celui-ci et les méthodes de développement utilisées dans notre projet.

III.2.1.2 La phase de conception

La phase de conception est une activité de modélisation qui est orientée non pas vers le monde mais vers le système. Elle consiste à modéliser le système à partir des modèles définis dans la phase d'analyse. Le modèle produit à l'issue de cette phase est le ***modèle de conception***. Celui-ci décrit la structure du système qui doit être construit en termes de mécanismes informatiques, de techniques de représentation et de logiciels nécessaires à l'implémentation des modèles de communication et d'expertise [Schreiber *et al.* 94]. Ainsi pour sa construction, le concepteur prend en compte les conditions externes même si l'élaboration demeure encore indépendante des soucis d'implémentation du système. En effet, le processus de conception est caractérisé par une succession de décisions [Van De Velde 94]. Celles-ci doivent d'abord être prises en accord avec l'architecture globale du système, et ensuite avec la sélection de techniques informatiques appropriées. Le processus de conception dans la méthode KADS se décompose en trois étapes [Schreiber *et al.* 93, 94]. La première consiste à faire le choix d'un paradigme d'architecture. En effet, généralement on distingue deux types de paradigmes d'architecture, l'architecture fonctionnelle et l'architecture orientée objet [Schreiber *et al.* 94]. Elles se distinguent de par les entités manipulées et les relations définies, c'est-à-dire par leur approche de décomposition. Une fois le paradigme d'architecture choisi, il faut procéder à la spécification de l'architecture. Pour cela, le concepteur doit utiliser le modèle conceptuel. En effet, celui-ci fournit l'architecture squelettique du système [Van de Velde 94], et la phase de conception va ajouter à cette architecture les aspects techniques, les outils de développement, les langages de programmation utilisés, etc. déterminés par la phase de conception. Ainsi, la deuxième étape consiste à prendre en compte les aspects informatiques, c'est-à-dire que le concepteur doit choisir les techniques informatiques qui vont être utilisées. Ce sont donc les techniques de résolution de problème, et dans le cas des SBC elles correspondent aux paradigmes de l'IA, parmi lesquels on retrouve les systèmes de production, la classification, la déduction automatique, etc. La troisième étape est relative aux choix d'un environnement logiciel. Celui-ci correspond à la détermination des langages d'IA et des environnements qui vont être utilisés pour l'implémentation. La phase de conception repose donc bien sur la spécification d'une architecture et sur la sélection de représentations et de techniques informatiques

appropriées. Si ce choix est généralement laissé libre au concepteur du moment que celui-ci respecte les conditions du modèle d'expertise, la méthodologie CommonKADS suit une approche qui consiste à préserver la structure du modèle conceptuel. Cela signifie que le contenu et la structure de l'information présents dans le modèle de niveau des connaissances sont conservés dans le produit final les rendant encore accessibles. Le fait d'utiliser cette approche offre de nombreux avantages notamment pour l'atteinte des objectifs cités pour la réalisation de SBC2, faisant particulièrement appel à la réutilisabilité, la maintenance, l'acquisition des connaissances par modélisation et enfin la génération d'explications. En effet, le fait de préserver la structure permet de conserver un accès aux différents niveaux de connaissance du modèle d'expertise. Or, cette connaissance est nécessaire à l'élaboration d'explications à différents niveaux d'abstraction et dans un langage qui soit compréhensible.

Une fois la phase de conception terminée, la phase d'implémentation peut alors commencer, nous n'allons pas décrire les dernières phases de la méthodologie KADS, celles-ci ayant été énumérées au début de la section III.2. Nous avons pu constater lors de la description de la méthode et plus particulièrement dans CommonKADS que tous les modèles développés traitent des interactions personne-machine. Ceci est directement lié à la génération d'explications qui, par définition, permettent d'établir une communication entre le système et l'application, ainsi qu'avec l'utilisateur. C'est pourquoi nous allons dans la partie suivante considérer la génération d'explications par rapport à la méthode KADS. Étant donné que certains aspects ont déjà été cités lors de la description de la méthode, nous veillerons à établir plutôt de nouvelles considérations concernant le sujet.

III.2.2 KADS et les explications

La méthode CommonKADS se distingue donc de KADS-I par l'intérêt particulier qui est apporté à l'interaction personne-machine. Plus précisément, plusieurs modèles traitent explicitement de la modélisation des explications ou tout du moins visent à la faciliter. C'est le cas des modèles d'agents et de communication, inexistants dans KADS-I, et qui permettent de représenter le modèle de l'utilisateur et les échanges d'informations avec le système. L'analyse des échanges d'information est en fait liée à l'analyse des explications qui devront être générées par le système. De même, l'analyse du modèle de l'utilisateur permet d'identifier le niveau d'abstraction et de complexité que le système final devra comporter. De plus, l'approche de conception, en préservant la structure du modèle conceptuel, permet de

faciliter, comme nous l'avons mentionné plus tôt, la génération d'explications car elle permet d'accéder à la connaissance modélisée dans le module d'expertise ainsi qu'aux liaisons entre le modèle de la connaissance et le code. En effet, d'après Möller [Möller 95], dont les travaux portent sur l'utilisation des modules de graphes conceptuels pour rendre opérationnels les modèles de KADS, pour avoir des SBC explicables, il est nécessaire d'avoir un modèle conceptuel explicite et déclaratif. Ainsi, la méthode CommonKADS semble être appropriée pour traiter les explications, c'est pourquoi nous retrouvons son utilisation dans de nombreux ouvrages liés aux explications, par exemple dans [Bourcier *et al.* 94; Sprenger 93; Aamodt 93; Martin 94]. Il est raisonnable de supposer qu'une méthode comme KADS, qui facilite le processus d'acquisition des connaissances facilitera aussi le processus de génération d'explications [Sprenger 93]. En effet, on a constaté que les méthodes de l'ingénierie de la connaissance à base de modèles comme KADS recommandent une approche au niveau des connaissances pour les explications [Baumewerd-Ahlmann *et al.* 91b]. L'approche développée par Sprenger, et qui constitue le projet DIAMOD, est de développer un composant pour générer des explications en langage naturel pour des systèmes experts à base de modèles qui utilisent l'approche KADS. Pour cela, il réalise l'étude des types d'explications à modéliser en fonction des niveaux de connaissance du modèle d'expertise. C'est le cas également dans les travaux de Martin [Martin 94] qui effectue la modélisation des explications au niveau du modèle d'expertise. Son approche est différente cependant, car il modifie la structure des modèles proposée dans KADS en intégrant un modèle d'expertise spécifique aux connaissances d'explications afin de pouvoir traiter les tâches explicatives. Lui aussi reconnaît le rôle important du modèle de communication pour représenter les échanges d'information entre agents ainsi que la spécification d'un modèle de l'utilisateur pour générer des explications adaptées à leurs besoins. De même, Grimnes [Grimnes&Aamodt 96] s'inspire de la méthode CommonKADS pour réaliser l'analyse du domaine du diagnostic médical à un niveau des connaissances. Il utilise quant à lui le niveau du domaine du modèle d'expertise pour élaborer des explications sur la résolution de problème. L'accent a surtout été mis sur l'élaboration d'une ontologie de la base de connaissances qui soit la plus proche possible du langage des radiologues. Ainsi, comme nous l'avons constaté pour la génération d'explications dans les SBC-RPC, le type de tâche à résoudre par le système est un facteur influençant la tâche de génération des explications dans un SBC. Ainsi, les tâches de planification complexes, comme l'estimation de l'impact environnemental, nécessitent des

SBC pour justifier leurs résultats par des explications compréhensives [Baumewerd-Ahlmann *et al.* 91b]. L'approche développée ici est une approche à base de modèles qui permet d'anticiper les questions des utilisateurs et de concevoir des mécanismes de réponses même pendant l'étape d'ingénierie de la connaissance. Plusieurs types de questions sont identifiés et leur sémantique est reliée aux composants structurés de KADS.

Les explications représentent donc une entité à part entière de la méthode KADS et surtout de KADS-II. Elles sont prises en compte lors de la construction de tous les modèles de la méthode. Aussi, le fait d'avoir reconnu l'importance d'élaborer des modèles orientés vers les explications, comme le modèle d'agent ou le modèle de communication, montre bien que si les explications ne représentent pas forcément un processus indépendant du processus général de résolution de problème, elles constituent toutefois une tâche de résolution à part entière. C'est pourquoi il est nécessaire de prendre en compte leur modélisation dès le début de l'analyse et de la conception d'un système. La méthode KADS représente une solution intéressante d'une part par l'importance accordée à la phase d'acquisition des connaissances, essentielle à la génération de bonnes explications et d'autre par parce qu'elle permet également du fait de son principe de modélisation de procéder à une analyse structurée des besoins en explications.

Ce chapitre nous a donc permis, pour commencer, de nous familiariser avec les différentes approches sur la génération d'explications dans les SBC-RPC. Celles-ci représentent la source d'inspiration sur laquelle nous avons basé nos propres travaux. La suite du chapitre consiste en la présentation de KADS, une méthodologie de développement de SBC en insistant surtout sur les propriétés de KADS-II, celle-ci s'avérant être plus adaptée à notre recherche qui consiste à modéliser la génération d'explications dans les SBC-RPC. Aussi, dans chacune des parties, nous avons réalisé en parallèle une étude des travaux effectués par les chercheurs du domaine sur les explications. Ceci nous a permis de constater que la génération d'explications est un sujet omniprésent dans la recherche en IA, mais qu'elle ne représente également qu'une partie des problèmes abordés. Ainsi, il est rare de trouver des travaux entièrement consacrés sur le sujet tout du moins de notre point de vue qui consiste à ne pas considérer les explications comme un outil permettant d'améliorer les performances d'un système de RPC mais plutôt comme un outil orienté vers l'utilisateur. C'est pourquoi, nous

allons dans le chapitre suivant décrire de manière plus précise les explications en reprenant et en complétant un modèle générique d'explications que nous avons élaboré dans notre mémoire de DEA [Verrière 97], et qui est donc rattaché à notre synthèse bibliographique dans notre travail de recherche de maîtrise.

CHAPITRE IV

Proposition d'un modèle générique d'explications

Les chapitres II et III nous ont permis de situer le contexte de notre recherche. En effet, nous avons décrit l'environnement des SBC, des SBC-RPC et de la méthodologie KADS. De plus nous avons introduit pour chacun d'eux l'importance qu'y tiennent les explications. Nous allons donc maintenant procéder à une analyse plus approfondie des explications. Pour cela, après avoir décrit les besoins en explications liés à l'élaboration d'un modèle générique, nous reprendrons les éléments du modèle générique d'explications que nous avons élaboré pour notre recherche de DEA [Verrière 97], où l'on trouve les principales références, en y intégrant les modifications apportées suite à l'avancement de nos travaux sur la génération d'explications.

IV.1 De l'étude des besoins en explications à l'élaboration d'un modèle générique

Pour commencer l'étude des explications, il est nécessaire de bien définir ce qu'est une explication. Nous avons déjà, dans les chapitres II et III, déterminé l'utilité et les différentes techniques ou approches élaborées pour générer des explications, mais nous n'avons pas, d'un point de vue autre que celui du développement logiciel informatique ou orienté utilisateur, défini le concept même. L'étude des explications est généralement réalisée dans les milieux littéraires comme pour les explications de texte par exemple. Il s'agit de faire une étude d'un texte afin d'en extraire les éléments qui nécessitent une réflexion particulière pour permettre de mieux comprendre dans son ensemble le sens du texte. Dans la réalité aussi, expliquer une situation revient à fournir les éléments qui permettent de mieux la comprendre. La génération d'explications en informatique n'étant pas différente, son rôle

consiste à apporter des informations qui permettront à leurs destinataires de comprendre les points obscurs auxquels ils se sont heurtés et qui peuvent les bloquer pour continuer leur tâche. Cependant, si le processus lorsqu'il est exécuté dans la vie courante se fait inconsciemment, il n'en est pas de même lorsqu'il s'agit d'en doter un système informatique. Ainsi, une question, que peut se poser à l'occasion une personne confrontée à un cas particulièrement délicat ou difficile à résoudre, doit être systématiquement prise en compte quand il s'agit de produire un dispositif explicatif : " Comment vais-je bien pouvoir expliquer ça ? " Cette question, pourtant simple en apparence, pour être résolue nécessite une analyse complète. En effet, son étude nous amène à explorer un ensemble de critères qui au début ne nous paraissent pas importants ni même existants. C'est l'étude de ces critères qui nous permet d'élaborer un modèle dit général d'explications [Verrière 97]. Ensuite, nous nous plaçons dans un ensemble de contextes variés où une explication est requise et, pour chacune des situations, nous analysons quels sont les éléments importants à prendre en compte, les concepts à définir, les raisonnements à justifier, etc. Cette étude des besoins en explications ajoutée à celle des critères entrant en jeu permet de former le modèle générique d'explications. Ainsi, pour élaborer le modèle il est nécessaire de se poser un ensemble de questions. La réponse à chacune de ces questions représente l'étude d'un critère spécifique nécessaire à la génération d'explications. Une étude bibliographique des travaux réalisés sur la typologie des explications et la génération des explications, dans les SBC tout d'abord puis dans les SBC-RPC en général, nous a permis d'identifier huit critères [Verrière *et al. 97*]. Il faut toutefois noter que notre modèle s'applique à n'importe quel type de système. En effet, l'objectif était d'élaborer un modèle suffisamment abstrait et générique pour pouvoir être instancié dans toute application nécessitant un dispositif explicatif.

IV.2 Étude des critères du modèle générique d'explications

Nous allons donc dans les paragraphes suivants définir les critères qui composent le modèle générique d'explications. Pour chacun d'eux, nous allons donner les principales caractéristiques à prendre en compte ainsi que le rôle qu'ils tiennent à l'intérieur du modèle.

IV.2.1 Quelle explication donner ?

Le premier critère correspond donc à ce que nous appellerons la typologie des explications Il a pour but de déterminer les connaissances explicatives. Il a fait l'objet de nombreux travaux et correspond à ce que les auteurs appellent : la typologie des questions [Chandrasekaran 89;

Baumewerd-Alhmann *et al.* 90]. Ce critère, le plus important de tous, est celui qui est le plus particulièrement adapté à une analyse selon KADS. C'est pourquoi, nous retrouverons son analyse détaillée dans la phase d'analyse qui fait l'objet du chapitre V. L'étude du type des explications est le plus souvent réalisée pendant la phase de modélisation du modèle d'expertise [Sprenger 93; Martin 94; Baumewerd-Ahlmann *et al.* 91a; Scott&Weckert 97; Dhaliwal 98]. En effet, le type d'explication à fournir est lié aux types de connaissances que l'on veut manipuler, il se situe donc au niveau de la connaissance, caractéristique de la méthode KADS.

La détermination des types d'explication est obtenue en faisant l'analyse des différentes questions qui peuvent être posées au cours de l'exécution d'un système. Celles-ci peuvent provenir de l'utilisateur mais aussi du système même. De manière générale, nous distinguons deux types d'explication, soit l'explication des connaissances du domaine et celle portant sur le raisonnement [Bourcier *et al.* 94].

IV.2.1.1 L'explication des connaissances

Ce type d'explication permet de fournir des informations sur la connaissance du domaine [Sprenger 93; Martin 94; Dhaliwal 98], ce que nous avons nommé "connaissance statique" dans le chapitre précédent. On peut ainsi entre autres, décrire le domaine d'application en énonçant par exemple les fonctionnalités du système, définir les concepts ou le vocabulaire utilisé, énoncer les règles, les processus utilisés, ou encore expliquer la terminologie utilisée par l'ingénieur de la connaissance pour implémenter le système. Elles sont donc toutes liées à la connaissance générale et représentent ce que l'on nomme des explications statiques [Scott&Weckert 97]. En effet, elles sont implantées dès la conception du système et sont conservées d'une exécution à l'autre. Ce sont les plus faciles à implémenter, néanmoins elles nécessitent une mise à jour parallèle avec les changements apportés au système. Elles demandent une analyse détaillée des connaissances du domaine ainsi qu'une représentation explicite de celles-ci. Elles peuvent être demandées à n'importe quel moment au cours du processus de résolution. Elles permettent de répondre aux questions de type *QUOI* [Bourcier *et al.* 94]. Par exemple, une question de ce type pourrait être : "Qu'est ce qu'une plante Liliacées ?", ou "À quelle variété de légumes appartient la citrouille ?".

IV.2.1.2 L'explication du raisonnement

Ce type d'explication repose sur les connaissances qui ont permis de trouver une solution. Ainsi, son but consiste à fournir à l'utilisateur les éléments qui lui permettent de comprendre le processus de résolution qui a été suivi par le résolveur de problème. Ces explications peuvent être considérées selon plusieurs types [Bourcier *et al.* 94; Baumewerd-Alhmann *et al.* 91b; Dhaliwal 98; Martin 94]. Elles peuvent soit expliquer simplement les résultats trouvés, on les met alors dans la catégorie des explications causales [Leake 95]. Elles ne font qu'accéder aux faits de la BC qui sont associés aux solutions. Elles permettent généralement de répondre aux questions de type *POURQUOI*. Un exemple de ce type de question serait : "Pourquoi avoir mis un plant de carotte à côté d'un plant de tomate ?". La réponse fait juste appel à des relations de compagnonnages spécifiant que la carotte et la tomate sont deux plantes amies, donc qu'elles peuvent être semées l'une à coté de l'autre. Le deuxième type consiste à élaborer des explications qui fournissent une trace du raisonnement fidèle qui a été suivi par le système comme celles développées dans MYCIN [Moore&Swartout 88; Gréboval 92]. Ces explications ne sont généralement utiles qu'aux concepteurs des systèmes car elles sont proches du langage d'implémentation et donc peu adaptées à un utilisateur non informaticien. La question type associée est *COMMENT*. Un exemple de ce type de question pourrait être "Comment le système est-il arrivé au résultat ?". La réponse liste les règles déclenchées dans l'ordre chronologique ainsi que les faits connus ou déduits. Le troisième type contient des explications justifiant les résultats [Goel *et al.* 96a]. La nuance entre les types 1 et 3 réside dans la nécessité d'accéder aux stratégies de résolutions de problème de la seconde [Sprenger 93]. Elle fait appel à des connaissances diverses qui sont à la fois liées aux processus de résolution mais aussi aux connaissances explicatives. Ces explications sont les plus difficiles à élaborer. Comme les explications sur la trace du raisonnement elles permettent de répondre à la question *COMMENT*. Un exemple de ce type serait "Comment le système a-t-il classifié le légume ?". La réponse spécifie les étapes de résolution et les connaissances impliquées, et elle ne respecte pas forcément l'ordre chronologique, à savoir que si la classification du légume a échoué à un moment donné, cela n'est pas mentionné dans l'explication finale. De plus, les explications de type 2 et 3 génèrent toutes deux des explications dynamiques [Scott&Weckert 97], c'est-à-dire des explications générées pendant le processus de résolution du système. Finalement, le dernier type d'explication permet de justifier pourquoi une solution n'a pas été trouvée [Bourcier *et al.* 94]. Ce type d'explication

est rarement pris en compte, mais il permet toutefois de vérifier la cohérence d'une solution par rapport à une autre. Il permet de répondre à la question *POURQUOI PAS*. Un exemple de ce type de question consiste en "Pourquoi ne pas avoir classifié le légume considéré dans la famille des ombelliféracées ?". Ici plusieurs techniques, dépendamment du sens de la question, peuvent être considérés. Il peut s'agir d'énoncer les faits qui ne sont pas vérifiés pour conclure à ce diagnostic ou de reprendre la trace du raisonnement, si l'on ne peut trouver le fait manquant directement et de chercher dans celle-ci la source d'erreur.

Ainsi la typologie des explications est déterminée par des questions types. Celles-ci s'adressent soit aux connaissances du domaine, soit au raisonnement suivi. L'application de cette typologie sera réalisée au chapitre VII dans le cadre des SBC-RPC, car elle est tout à fait appropriée à une analyse selon la méthodologie KADS. La typologie représente donc le premier critère à prendre en compte de notre modèle, cependant, nous pouvons constater d'après ce que nous avons décrit que le type d'explication est déterminé par d'autres facteurs. En effet, le type de l'utilisateur de l'explication est généralement lié au type d'explication dont le système a besoin. C'est pourquoi, le type de l'utilisateur représente le deuxième critère du modèle générique.

IV.2.2 À qui expliquer ?

Ce critère a donc pour objectif de déterminer quels sont les bénéficiaires des explications. En effet, on peut distinguer plusieurs types d'utilisateurs [Sprenger 93; Martin 94]. Tout d'abord il faut déterminer si le destinataire de l'explication est un humain ou non. En effet, comme nous l'avons précisé lors de l'étude des SBC-RPC, les explications représentent souvent un outil utilisé par le système pour améliorer ses performances [Bento *et al.* 94a; Macedo *et al.* 96]. Aussi, les techniques explicatives ne seront pas les mêmes si elles sont utilisées dans le processus de résolution d'un système. Les types d'utilisateurs sont déterminés dans KADS-II lors de l'étude du modèle d'agents [Wærn&Gala 93]. Celui-ci contient la description plus ou moins détaillée des agents intervenant au cours de la résolution de problème. Ainsi, KADS-II permet d'élaborer autant de modèles d'agent qu'il y a d'agents effectuant des tâches différentes. Aussi, il distingue le modèle du système du modèle de l'utilisateur. Le modèle qui nous intéresse plus particulièrement ici est le modèle de l'utilisateur tel que décrit dans [Martin 94]. En effet, l'explication à élaborer ne sera pas la même suivant le type de l'utilisateur du système. Il faut faire des distinctions entre les différents niveaux de

connaissance et de compétence de celui-ci [Dhaliwal 98]. Ainsi on distingue trois types d'utilisateur [Martin 94, Sprenger 93] : l'expert du domaine, l'ingénieur de la connaissance et l'utilisateur non expérimenté.

IV.2.2.1 L'expert du domaine

C'est celui qui détient la connaissance concernant le domaine d'application considéré. C'est sur ses connaissances et ses qualités à expliciter les processus de raisonnement tels qu'ils sont suivis dans la réalité que repose la phase d'acquisition des connaissances indispensable à l'élaboration de tout SBC. Aussi, lorsqu'un système est terminé, l'expert peut avoir besoin d'explications. Celles-ci sont relatives à la pertinence des résultats retrouvés. En effet, s'il s'agit d'un diagnostic médical par exemple, l'expert va vouloir vérifier et valider le diagnostic trouvé par le système. Pour cela, il va comparer les hypothèses prises en compte par l'application avec les siennes, et ainsi déterminer si le raisonnement suivi par le système s'accorde bien au sien. Il peut aussi vouloir comprendre pourquoi le résultat trouvé est erroné, à cette fin il va demander sur quels faits repose le diagnostic afin de trouver les erreurs de raisonnement. Ainsi, l'expert demande des explications lui permettant de valider un système en vérifiant que ce dernier imite de manière suffisamment fidèle son raisonnement pour aboutir aux mêmes résultats. Une autre catégorie d'experts peut aussi avoir besoin d'explications. En effet, le rôle des SBC consiste à concentrer l'expertise pour qu'elle soit réutilisable à long terme. Aussi, il arrive que des experts utilisent des SBC pour lesquels ils ne partagent pas les connaissances d'expertise, les explications leur permettent alors de comprendre d'autres méthodes de raisonnement dont ils pourront alors se servir pour leur propre analyse. C'est ce que nous avons pu conclure de notre expérimentation dans le jardinage en carrés en utilisant Easy Reasoner.

IV.2.2.2 L'ingénieur de la connaissance

Son rôle consiste à interpréter la connaissance qu'il a recueillie de l'expert afin de la modéliser et de la représenter sous une forme qui soit utilisable dans une machine. Les explications dont il a besoin portent sur la validité de cette transformation des connaissances. En effet, à la différence de l'expert, il ne vérifie pas l'exactitude des résultats obtenus mais plutôt la technique de résolution implantée. Les explications lui permettent de comprendre les erreurs de conception qui ont pu être faites. Ces explications peuvent correspondre tout simplement à une trace du raisonnement en langage machine, par exemple la liste des règles déclenchées

en langage Prolog. Ainsi, les explications dont l'ingénieur de la connaissance a besoin portent sur la résolution de problème suivie par le système, et elles permettent d'en tester la robustesse et la qualité.

IV.2.2.3 L'utilisateur non expérimenté

C'est le dernier type d'utilisateur. Tout d'abord on retrouve dans cette catégorie l'utilisateur non informaticien qui a besoin d'explications pour se familiariser avec l'environnement informatique. On peut par exemple se figurer un utilisateur demandant une aide pour lui expliquer les procédures informatiques à suivre, comme un mode d'emploi. La deuxième catégorie concerne les utilisateurs qui se servent d'un système d'apprentissage. Les explications vont alors tenir un rôle d'aide à la décision, et représentent un outil d'enseignement. Ce sont donc les utilisateurs qui ne sont pas experts du domaine et qui ont besoin d'explications pour apprendre les connaissances relatives au domaine d'application. Ce type d'utilisateur a donc besoin d'explications pour apprendre et en quelque sorte devenir expert en la matière.

Ainsi, une fois le type d'utilisateur déterminé il faut faire le bilan de ses caractéristiques, centre d'intérêt, croyances, niveau de compétence, préférences, niveau de langage, etc. [Greer *et al.* 94]. Le processus de génération d'explications devra tenir compte de ces facteurs afin de produire des explications plus adaptées aux besoins. Ceci évite de produire des explications qui soient trop lourdes ou insuffisantes aux yeux de leur destinataire. De plus, cela amène à collecter un certain type de connaissances dès la phase d'acquisition des connaissances même si celles-ci ne sont pas forcément utiles au processus de résolution aux yeux des experts. En effet, elles peuvent s'avérer indispensables à la bonne compréhension des utilisateurs. Il en est de même lorsque le système est le destinataire des explications. L'étude de ses besoins, du rôle que les explications tiennent dans son évolution, des dépendances du système envers d'autres agents, et des tâches qu'il doit réaliser permet de définir un cadre d'analyse qui sera pris en compte lors de l'élaboration des explications. Nous pouvons noter ici que l'étude des utilisateurs potentiels des explications doit se faire suffisamment tôt lors du processus de développement d'un système pour pouvoir être prise en compte dès les phases d'analyse des tâches, de stratégies et de conception du système. Mais on peut également constater que l'étude du type du destinataire des explications permet de déterminer la représentation que doivent adopter les explications afin d'être appropriées.

IV.2.3 Sous quelle forme expliquer ?

Comme nous l'avons émis au paragraphe précédent, l'ingénieur de la connaissance pourra se satisfaire d'une trace des règles déclenchées lors de la résolution de problème alors que l'utilisateur non expérimenté désirera plutôt avoir des écrans d'aide comme explications. Le mode de présentation des explications est donc un critère à considérer lors de la génération d'explications. On distingue deux types de représentation : la représentation interne et la représentation externe.

IV.2.3.1 La représentation externe

Elle correspond à ce que l'utilisateur qui demande une explication voit quand celle-ci lui apparaît, et plus précisément sous quelle forme. Ainsi, les présentations les plus fréquentes sont les graphismes [Scott&Weckert 97], les tableaux car ceux-ci permettent de condenser le maximum d'information de manière logique par rapport aux concepts et relations manipulés ou encore les images qui permettent de représenter les connaissances de manière explicite. Cependant, c'est l'utilisation des textes, du dialogue ou du langage naturel qui présentent à l'heure actuelle le plus d'intérêt [Bourcier&Gréboval 92]. En effet, les explications produites sont alors plus facilement compréhensibles car exprimées à un niveau de langage accessible à l'utilisateur. Il faut bien sûr faire une distinction entre les représentations sous forme de textes qui sont généralement des structures prédéfinies à trous qui sont instanciées sur demande avec les valeurs courantes et qui font partie des explications statiques, des représentations en langage naturel qui sont élaborées de manière dynamique. Cependant, et comme nous le mentionnons plus tôt, une simple liste de faits et règles déclenchées, appelée trace de règles, dans un langage machine peut suffire et est même souhaitable si l'utilisateur est familier avec le système. Il est donc nécessaire de prendre en compte la nature de l'utilisation de l'explication et de son destinataire. Il faut noter que la décision d'utiliser telle ou telle représentation est, dans la méthodologie KADS, déterminée lors de l'étude du modèle de communication qui porte sur les échanges personne-machine. C'est là qu'est pris en compte l'essentiel des besoins ergonomiques d'une application, avec la détermination des interfaces. Cependant, les outils et la représentation exacte ne sont pas encore déterminés.

IV.2.3.2 La représentation interne

Elle est plus complexe et correspond, sauf pour le dernier cas de figure présenté, à ce que l'on ne voit pas. C'est le formalisme utilisé pour représenter les explications. La décision d'opter pour tel ou tel formalisme n'est prise qu'assez tard dans le déroulement de la méthodologie KADS-II. Aussi, elle ne devrait intervenir que lors de la phase de conception même si elle est déjà plus ou moins définie lors du choix d'une technique de résolution. Ainsi, si le raisonnement utilisé est à base de règles, il paraît naturel d'envisager de représenter les explications dans la logique du premier ordre avec l'utilisation de prédicats. Si l'on opte par contre pour un raisonnement orienté objet, les explications seront alors certainement des objets. Ainsi, il est usuel lorsqu'on utilise le RPC d'insérer les explications directement à l'intérieur des cas. Les explications sont donc alors représentées selon le même formalisme que les cas [Leake 95; Aamodt 93; Goel *et al.* 96b]. Cependant il ne faut pas faire ces déductions de choix de formalisme systématiquement, la décision reposant avant tout sur le type d'explication à produire. On peut dans un même système avoir besoin d'explications sous forme de patrons d'explications, tels qu'ils sont définis par [Schank *et al.* 94], de frames, de schémas, de scripts, de réseaux sémantiques, d'arbres de décision qui peuvent être représentés par des objets ou même des listes chaînées, ou encore utiliser des graphes conceptuels [Möller 95], et cela même si le système n'est pas un système hybride. La décision dépend des tâches effectuées à des moments particuliers du raisonnement.

Le choix d'un formalisme de représentation des explications, qu'il soit d'un niveau externe (présentation) ou d'un niveau interne (représentation), représente donc un critère important à prendre en compte pour la génération d'explications. Ceci permet d'introduire le critère suivant qui consiste à déterminer à quel moment des explications doivent être produites.

IV.2.4 Quand expliquer ?

Ce critère comme les précédents peut être considéré selon deux points de vue. En effet, lorsqu'on génère des explications, il faut se placer aux niveaux de l'utilisateur et du système. Le premier concerne le moment auquel les explications vont être fournies à l'utilisateur. Il est donc nécessaire d'identifier à quelles étapes de la résolution de problèmes l'utilisateur peut en faire la demande. Le deuxième concerne le moment auquel le système doit déclencher le processus de construction des explications.

IV.2.4.1 Quand fournir une explication ?

L'exécution d'un SBC respecte plusieurs étapes de résolution. Aussi, l'utilisateur peut, à tout moment, avoir besoin d'éclaircissements. Évidemment, le type d'explications produit suit le processus de résolution et est donc déterminé en fonction des phases de développement. Ainsi, la plupart des explications sont fournies à l'issue d'une résolution quand les résultats finaux sont produits. En effet, l'utilisateur peut avoir besoin de se faire expliquer ces résultats, il désire alors en connaître la pertinence afin de les accepter, ou en avoir une justification s'il ne les comprend pas. En outre, l'utilisateur peut à tout moment s'interroger sur un concept ou une terminologie qu'il ne maîtrise pas, les définitions demandées ou les aides à montrer doivent alors être accessibles même si elles n'ont pas de rapport avec l'étape en cours. Finalement, lorsqu'il s'agit de raisonnement particulier comme le RPC par exemple qui suit un cycle de résolution, à chaque terminaison d'une phase du cycle, l'utilisateur peut demander une explication relative au raisonnement spécifique de la phase ou aux résultats obtenus. La particularité vient ici du fait que lorsque l'utilisateur approuve les résultats intermédiaires et déclenche la phase suivante du processus de résolution, les explications fournies pour justifier l'étape suivante ne permettront pas, en général, de revenir sur les explications produites antérieurement. Ainsi, une explication peut être nécessaire en début d'exécution pour présenter une application par exemple, pendant l'exécution pour justifier les résultats intermédiaires et à la fin pour valider les résultats obtenus.

IV.2.4.2 Quand produire une explication ?

Cet aspect doit être pris en compte, selon la méthodologie KADS-II, dès l'élaboration des modèles de tâches et de communication, et au niveau stratégique du modèle d'expertise si l'on veut produire des explications les plus complètes possibles. En effet, cela permet lors de la conception du système de prendre en compte la génération d'explications, et de l'intégrer à la résolution de problèmes. Ainsi, suivant le type d'explication, l'élaboration d'explications peut être tout à fait indépendante du processus de résolution, c'est le cas par exemple des explications sur les connaissances du domaine, ou en être dépendante. Il faut alors anticiper les éventuelles interruptions des utilisateurs et pour chaque étape pertinente de résolution récupérer les connaissances explicatives nécessaires à leur élaboration. La distinction entre les deux permet d'élaborer des explications statiques ou dynamiques. Les premières étant micro programmées (*hard-coded*), elles survivent aux arrêts du système, tandis que les secondes

étant volatiles, elles sont développées quand le système tourne, sont effacées de la mémoire à chaque résolution et sont dépendantes du processus de résolution. La détermination de l'une ou l'autre repose également sur le but que l'on veut atteindre lorsqu'on génère une explication. L'étude de ce but fait l'objet du critère suivant.

IV.2.5 Pourquoi expliquer ?

L'identification des types d'explication que l'on veut produire repose en fait sur la volonté d'atteindre un but qui satisfasse l'utilisateur. Ainsi, on peut identifier un ensemble varié de buts qui motivent la génération d'explications. Ceux-ci se situent à un niveau d'abstraction inférieur de celui contenant les types d'explication. Ainsi, pour un même type d'explication on peut référer à plusieurs buts d'explications. Par exemple, des explications sur le raisonnement peuvent être produites pour persuader l'utilisateur du résultat obtenu ou tout simplement pour lui apprendre comment le processus de résolution est suivi. Pour déterminer le but d'une explication, il faut donc deviner ce qu'attend l'utilisateur de celle-ci. Pour cela, il faut analyser la tâche de résolution de problème et considérer dans quelle mesure celle-ci et les connaissances qu'elle manipule peuvent poser problème à l'utilisateur. De plus, nous avons déterminé que le système aussi pouvait avoir besoin d'explications, étudier dans quel but représente donc un aspect important pour le traitement des explications.

Ainsi, nous pouvons identifier les buts d'explications suivants [Martin 94]:

- **explorer** les connaissances du domaine ou fournir de l'information ;
- **vérifier** la cohérence des données manipulées et du raisonnement implémenté ;
- **valider** la prise de décision du système pendant la phase de résolution de problèmes ;
- **évaluer** les résultats ;
- **former** les experts ou les débutants dans leur discipline ;
- **informer** l'utilisateur sur les actions passées ou futures du système pour augmenter sa confiance ;
- **justifier** le comportement, et tout ce qui a attrait au raisonnement suivi et aux résultats trouvés ;
- **persuader** l'utilisateur de la cohérence et de la validité des résultats même s'il n'est pas d'accord avec le raisonnement suivi.

L'étude de ce critère ne suit pas une analyse aussi détaillée que celle des autres critères. Il agit davantage à titre informatif dans la génération d'explications. En effet, les termes vérifier et valider sont différents, cependant on s'accorde également à conclure que tous deux doivent produire une justification établissant que les résultats sont corrects. La prise en compte de ce critère a donc pour rôle de modifier l'état d'esprit dans lequel le concepteur se place et à l'obliger à prendre en compte des connaissances supplémentaires afin d'apporter des informations qui satisferont le but initialement suivi. S'il faut construire une explication dont le but consiste à convaincre l'utilisateur, il faudra alors utiliser des arguments adaptés et les choisir en fonction du niveau de connaissance de l'utilisateur. L'étude des buts d'explications permet donc de montrer l'importance et l'utilité du modèle de l'utilisateur, et permet d'identifier sur quelles connaissances bâtir l'explication. Ainsi, considérer pourquoi une explication est demandée permet d'une part d'identifier les connaissances à utiliser et d'autre part de déterminer les types d'explication qui devront être générés. Mais l'étude de ce critère entraîne également celle des raisons qui ont stimulé l'utilisateur dans sa requête. L'objet de l'explication représente le critère suivant du modèle.

IV.2.6 Quoi expliquer ?

Lorsqu'un utilisateur fait une requête d'explication, c'est qu'il est confronté à un problème. L'identification de la nature de ce problème représente un facteur déterminant lors de l'étude des explications car il permet d'identifier l'objet et les structures associées sur lesquels devra se porter l'analyse. Il existe dans notre environnement une quantité indénombrable de raisons qui vont nous amener à un moment ou un autre à nous poser des questions. Ainsi, dans le domaine de la génération d'explications en informatique, la liste de ces éléments déclencheurs est restreinte et peut être facilement déduite de l'étude des critères précédents. Nous avons déterminé qu'une explication était nécessaire quand un utilisateur ne comprend pas les connaissances manipulées ou le raisonnement suivi. Ici, nous cherchons une description plus détaillée de ces connaissances ou de l'élément du raisonnement incompris. Pour l'étude de ce critère, nous devons distinguer deux cas. Le premier consiste à se placer dans une application ou les explications sont le noyau du raisonnement. C'est le cas par exemple de la plupart des travaux réalisés sur les explications dans les SBC-RPC qui reposent sur les explications d'anomalies de la vie courante [Leake 95; Aamodt 93]. Les applications ont pour but, à partir d'un énoncé, d'en extraire les faits qui semblent bizarres, et de les

expliquer à partir de situations passées similaires. Dans le système SWALE [Leake 95], le problème de base consistait à expliquer la mort d'un cheval. C'est un problème de la vie courante considéré comme une anomalie car a priori rien ne laissait présager que ce cheval allait mourir puisqu'il venait de sortir gagnant d'une grande course, qu'il était jeune et en bonne santé. Il s'agit donc d'expliquer un événement bizarre qui n'est pas en accord avec les connaissances acquises et ce, à partir des connaissances dont on dispose. Cependant d'autres objets liés à l'application et à son environnement peuvent être à l'origine d'un besoin d'explications, ce sont en général les plus courants. En effet, dans le deuxième cas, les explications ont pour but d'apporter un surplus d'information à celle déjà fournie à l'utilisateur. Celui-ci peut avoir un problème de compréhension vis-à-vis des connaissances manipulées, connaissances qui peuvent être liées au domaine comme la terminologie employée, les concepts et les relations, ou au raisonnement servant à la résolution de problème. Mais, le problème de compréhension peut aussi être lié aux résultats, s'ils s'avèrent surprenants ou inattendus. En effet, l'utilisateur peut être en désaccord avec la solution donnée par le système. L'explication portera donc sur celle-ci et les éléments qui ont permis de la déterminer. Ainsi, les hypothèses qui ont servi à construire la solution peuvent aussi représenter une source de problèmes. Il s'agit alors de vérifier si elles sont en accord avec les données du problème et la solution associée. Mais l'explication peut aussi porter sur les préférences à l'origine du choix de la stratégie de résolution. Celles-ci sont directement liées aux actions et recommandations du système qui peuvent à leur tour être mises en doute.

Ainsi, déterminer sur quoi va porter l'explication permet d'identifier les structures et les objets sur lesquels va devoir être construite l'explication. Il existe un lien étroit entre les critères quoi et pourquoi. En effet, il faut déterminer l'objet qui doit être expliqué en fonction du but désiré de l'explication, ceci permet d'identifier les types d'explication qui seront finalement générés. Nous venons de définir une certaine méthodologie explicative, en effet afin d'élaborer des explications il est nécessaire de choisir une méthode.

IV.2.7 Comment expliquer ?

Dans le modèle générique d'explications que nous avons décrit dans notre rapport de DEA, nous avions considéré deux axes d'études pour l'étude de ce critère [Verrière 97]. En effet, il faut distinguer la stratégie que l'on va adopter pour construire une explication, de la technique que l'on va utiliser pour l'implémenter.

IV.2.7.1 Les stratégies explicatives

Une stratégie explicative est un mécanisme qui peut être associatif, opportuniste, d'exploitation, prescriptif et restrictif, et qui permet de choisir et d'organiser le contenu d'une explication [Martin 94]. Elle a donc pour but de fixer les différentes manières dont l'information doit être incluse dans l'explication, d'établir la manière dont la connaissance doit être structurée et comment les structures ainsi réalisées vont interagir et à partir de quelles connaissances [Sprenger 93; Baumewerd-Hallman *et al.* 90]. Ainsi, le rôle d'une stratégie explicative est d'organiser la connaissance en fonction du but à atteindre. Elle consiste donc à trouver un moyen pour lier l'objet de l'explication (du critère *quoi expliquer*) au but d'explication (du critère *pourquoi expliquer*). Il faut noter que les stratégies explicatives ne sont pas traitées au niveau de l'implémentation des explications mais qu'elles sont déterminées lors de l'analyse. Elles font appel à l'ensemble des critères déjà définis tels que la forme externe des explications pour, par exemple, montrer à l'utilisateur par un graphique ou un tableau que les résultats obtenus sont corrects. Une stratégie explicative peut être une comparaison, une justification, un simple commentaire ou une définition, une critique, une évaluation. Chaque stratégie explicative est donc propre à une explication particulière. Elle dépend donc d'une analyse précise des besoins en explication d'une application en accord avec le domaine traité. Plus une application possède de stratégies explicatives, plus elle sera riche et plus les explications qu'elle produira seront diversifiées et adaptées aux besoins des utilisateurs. En effet, on peut même combiner plusieurs stratégies explicatives afin de donner plus d'impact à une explication si l'on veut vraiment persuader un utilisateur et le convaincre, et donc satisfaire le but d'explication que l'on s'est fixé.

IV.2.7.2 Les techniques explicatives

Les techniques d'explication consistent, quant à elles, à adopter une méthode permettant d'implémenter les explications. Cette méthode peut être indépendante des propriétés et caractéristiques définies dans les différents modèles déterminés lors de l'étude des critères du modèle. Ainsi, plusieurs approches ont été suivies dans les travaux réalisés par les chercheurs sur les explications. Il a longtemps été considéré que la génération d'explications devait être en quelque sorte greffée au processus de résolution d'une application. C'est pourquoi elle ne faisait pas l'objet d'une étude spécifique. L'insatisfaction rencontrée face aux explications générées, de par leur manque de clarté et de compréhensibilité, a mené les chercheurs à

considérer la génération d'explications comme une tâche de résolution à part entière, nécessitant sa propre analyse et l'utilisation de techniques et de stratégies explicatives particulières. Concernant l'explication des connaissances, les techniques d'explications reposent sur le choix d'un formalisme de représentation et d'une technique visant à fournir des explications qui soient les plus explicites possibles. Ainsi, les techniques d'explications concernent davantage les explications du raisonnement. La principale interrogation qui se pose concernant la production d'explications du raisonnement consiste à déterminer s'il faut ou non éloigner la ligne de raisonnement de la ligne explicative [Bourcier *et al.* 94]. En effet, lorsque le système résout un problème, il suit la ligne de raisonnement qui a été déterminée, à savoir dans le cas d'un SBR, il va, à partir d'un ensemble de faits initiaux, déclencher les règles dont les prémisses s'accordent avec ces faits et les nouveaux faits déduits, jusqu'à aboutir à une solution finale, dans le cas d'un SBC-RPC, le système va suivre fidèlement le cycle des quatre phases du RPC selon l'implémentation qui en a été faite. Cependant, cette ligne de raisonnement, si elle permet d'engendrer des solutions, et si elle peut être stockée et reproduite à l'utilisateur, n'offre pas un degré de compréhensibilité très satisfaisant et encore moins explicite. Dans la vie courante, lorsqu'on veut expliquer un phénomène on va généralement partir de la solution et en faisant le raisonnement inverse, revenir aux faits du problème. On évite ainsi d'explorer les branches du raisonnement qui n'ont pas abouti et qui ont été laissées de côté. Ainsi, si la ligne de raisonnement est réalisée selon le chaînage avant, la ligne explicative elle, suit un raisonnement en chaînage arrière. Cela permet généralement de limiter la quantité d'information que l'on introduira dans l'explication. De plus cela évite d'avoir à faire un filtrage important des informations superflues qui rendent incompréhensibles les explications. Ainsi, la question se pose là, faut-il expliquer selon la ligne explicative ou simplement reproduire le raisonnement tel qu'il a été suivi. La réponse en fait dépend des explications que l'on veut produire et à qui elles sont destinées. Ainsi, si le destinataire des explications est le concepteur, il est plus approprié de suivre le raisonnement du système car cela lui permettra de déceler les erreurs et de vérifier si son "algorithme général" est correct. Par contre, s'il s'agit d'un simple étudiant qui utilise d'un outil d'apprentissage, la ligne d'explication lui montrera exactement le raisonnement à suivre en faisant abstraction des mauvais choix qu'il pourrait prendre.

Ainsi, encore une fois, le choix de stratégies ou techniques explicatives est totalement dépendant des explications que l'on veut générer. Cependant, générer des explications en adoptant la stratégie de la ligne explicative demande un effort supplémentaire. En effet, afin de rendre les explications plus complètes, il va falloir faire appel à des connaissances et des informations qui sont certes utiles à l'utilisateur mais inutiles au système dans son processus de résolution. C'est pourquoi le dernier critère que nous considérerons concerne le type de connaissances à utiliser pour générer des explications.

IV.2.8 Avec quelles connaissances expliquer ?

Il ne suffit pas, pour produire des explications, d'avoir identifié leur environnement, à savoir les besoins qu'elles doivent satisfaire conformément aux caractéristiques des usagers et de l'application en général. En effet, il est nécessaire à ce stade de déterminer les connaissances qui vont permettre d'engendrer de telles explications. D'après Swartout, pour connaître les connaissances utiles qui doivent être intégrées dans le système pour fournir des explications, il suffit d'identifier les questions qui pourraient être posées [Swartout&Moore 93]. L'étude de ces questions est également utilisée pour déterminer les types d'explication. En effet, les types de connaissances à intégrer dépendent des types d'explication que l'on veut fournir. Cependant le problème concernant les connaissances à utiliser pour générer des explications n'est pas tant de les identifier mais plutôt de déterminer à quel niveau les modéliser.

Le problème ici est lié au conflit existant entre les connaissances explicatives et les connaissances de résolution de problème. En effet, il est important de représenter non seulement la connaissance dont le système a besoin pour accomplir ses tâches mais aussi les décisions prises au départ et qui sont dans la plupart des cas implicites parce qu'elles influencent les décisions du cogniticien mais ne sont plus disponibles dans le système final [Sprenger 93]. Aussi, faut-il séparer ces deux types de connaissances ? Pour répondre nous devons déterminer quelles sont les connaissances nécessaires à la résolution de problèmes et celles impliquées dans la génération d'explications.

Ainsi, pour résoudre un problème, un système utilise trois types de connaissances différentes [Sprenger 93] :

- **La connaissance du domaine** ou connaissance générale permet de décrire l'environnement ou, selon KADS, le contexte organisationnel. Elle est au cœur de n'importe

quel SBC et est basée sur les connaissances de l'expert du domaine. C'est la connaissance profonde [Scott&Weckert 97] qui est représentée dans les règles sous la forme de concepts et de relations. Elle peut être terminologique, descriptive du domaine, etc..

- **La connaissance d'inférence**, différente de la connaissance du domaine, est utilisée pour pouvoir raisonner sur le modèle du domaine et pour trouver des solutions aux tâches spécifiques.
- **La connaissance de contrôle** est utilisée pour guider les processus d'inférences, résoudre les problèmes pouvant survenir lors du raisonnement.

De son côté, la génération d'explications fait appel aux types de connaissances suivants [Scott&Weckert 97; Dhaliwal 98]:

- **La connaissance du domaine**, si elle est décrite de manière suffisamment explicite dès le départ, peut être la même que celle du processus de résolution.
- **La connaissance de la nature des explications** correspond à ce que nous avons défini au travers de l'étude des différents critères, à savoir les stratégies explicatives en fonction des types d'explication à produire et des utilisateurs, les connaissances sur comment structurer une explication, comment représenter et présenter une explication à différents niveaux de détail ou de difficulté.
- **La connaissance de l'utilisateur** qui est contenue dans le modèle de l'utilisateur. Cette connaissance est la plus difficile à obtenir et à maintenir, car elle change à mesure que l'utilisateur acquiert de nouvelles connaissances. D'elle dépend la qualité d'une explication.
- **La métaconnaissance** permet de clarifier la stratégie de résolution de problème et la structure de représentation des connaissances. C'est la connaissance stratégique.
- **La connaissance spécifique du cas** permet à un niveau moins abstrait de fournir des explications sur les données et les résultats spécifiques du cas.

Nous pouvons constater que les types de connaissances rappellent les connaissances modélisées dans le modèle d'expertise de KADS. Martin d'ailleurs dans ses travaux réalise deux modèles d'expertise distincts, le modèle général et un autre pour les explications [Martin 94]. En effet, plusieurs approches ont tenté de séparer les connaissances explicatives des connaissances utiles à la résolution de problème en construisant deux bases de connaissances distinctes. Ceci avait pour objectif de pouvoir faire une distinction nette entre

les deux. En effet, comme nous l'avons mentionné plus tôt, les connaissances liées aux décisions du système ne sont pas conservées dans la BC du système car elles sont inutiles pour la suite du processus, alors qu'elles sont utiles pour générer des explications. C'est pourquoi il faut les intégrer dans une autre base de connaissances explicatives. Le problème lié à la séparation des deux types de connaissances réside dans les difficultés de mise à jour. En effet, en créant deux bases de connaissance distinctes, certaines connaissances vont être dupliquées car elles sont utiles à la fois à la génération d'explications et à la résolution de problèmes. Aussi, si des modifications doivent être apportées à la BC du système, il faut en parallèle effectuer les changements nécessaires dans la base de connaissances explicatives. Ceci entraîne donc des problèmes de maintenance et représente à terme une source d'erreurs. Aussi, il paraît plus satisfaisant de modéliser de manière explicite les connaissances utilisables à la fois par le module de résolution de problème et le module explicatif afin de les rendre facilement accessibles aux deux. Ensuite, les connaissances spécifiques aux explications peuvent être modélisées séparément sachant que les explications, si elles représentent un processus de résolution à part entière sont souvent générées pendant le processus de résolution. Aussi, on ne peut considérer que les connaissances manipulées puissent être totalement indépendantes les unes des autres. Les deux bases de connaissances, générale et explicative, doivent être reliées l'une à l'autre afin de créer une communication permettant d'effectuer les modifications requises à tout moment. Finalement, la connaissance relative au cas courant est très importante pour la génération d'explications même si elle ne figure en général pas parmi les types de connaissances utiles. C'est elle qui engendre le processus de résolution, aussi c'est sur elle que repose notamment la justification causale des solutions. Ainsi, cette connaissance est très importante dans le RPC. En effet, elle sert de support aux modifications apportées suite au processus d'adaptation. En la conservant, on garde une trace du raisonnement ou des éléments qui ont permis de changer la solution du cas retrouvé. Ceci permet lors d'une autre résolution où ce cas est retrouvé de disposer de toute la connaissance qui a été utilisée pour aboutir à la solution. Cela permet, entre autre, d'éviter des erreurs.

Dans ce chapitre, nous avons donc décrit le modèle générique d'explications que nous avons élaboré. Celui-ci représente un modèle général contenant un ensemble de critères à prendre

en compte lorsqu'on veut concevoir un module explicatif. La transformation du modèle générique en module explicatif est obtenue en réalisant une instanciation des critères du modèle en accord avec l'application considérée. Ainsi, dans le cas de notre application de jardinage en carrés par exemple, pour doter le système résultant EDEN de capacité explicative il est nécessaire de définir les types d'explication que l'on veut produire, à qui elles vont bénéficier, à quel moment elles vont devoir être élaborées, sous quelle forme, dans quel but, pour quelles raisons et enfin quelles vont être les connaissances à modéliser pour y arriver. En effet, le type des connaissances à utiliser pour générer des explications est déterminant pour la qualité des explications. Néanmoins, celui-ci est souvent déterminé lors de l'étude des explications que l'on va générer. En effet, à un type d'explication correspond un type de connaissance à modéliser. C'est pourquoi l'étude des types d'explication doit faire l'objet d'une analyse plus approfondie. En effet, tous les critères que nous avons intégrés dans notre modèle générique d'explications ne peuvent vraiment être utiles que si l'on connaît les types d'explication que nous voulons générer. Ceux-ci sont obtenus suite à l'étude des besoins de l'application en explications. Nous avons choisi la méthode KADS pour procéder à cette étude. En effet, nous avons montré qu'elle était adaptée pour réaliser l'analyse des différents critères du modèle car ils peuvent être pris en compte lors de l'élaboration des différents modèles. Les critères que nous avons définis ne demandent pas tous le même effort d'analyse. En effet, certains comme la présentation des explications ou l'événement déclencheur du processus explicatif sont dépendants de l'application considérée. Ils nécessitent une connaissance du domaine d'application. Par contre d'autres critères tels que l'identification des types d'explication ou des types de connaissances peuvent être traités à un niveau d'abstraction plus élevé. En effet, nous étudions la génération des explications dans les SBC-RPC. Ainsi, nous nous intéressons plus particulièrement à la typologie des explications utiles dans ces systèmes. C'est pourquoi nous allons procéder à l'analyse des types d'explication selon KADS. Aussi, le chapitre VI portera sur la modélisation de la résolution de problème selon KADS de notre application de jardinage en carrés, et le chapitre VI portera lui sur la modélisation des explications et sur la phase de conception du système.

CHAPITRE V

Modélisation de la Résolution de Problème

Dans ce cinquième chapitre et dans le suivant, nous allons faire l'analyse de la tâche de génération d'explications dans les SBC-PRC en utilisant la méthode KADS. Nous avons, dans le chapitre précédent, présenté le modèle générique d'explications que nous avons élaboré. Celui-ci contient l'ensemble des critères à prendre en compte lors de la modélisation d'un module d'explications dans un système. Certains sont spécifiques au domaine d'application tandis que d'autres peuvent être analysés de manière indépendante de celui-ci. C'est pourquoi nous allons faire l'analyse KADS en nous plaçant dans le cadre général des SBC-RPC, tout en faisant référence à l'application de jardinage en carrés. Cette approche nous permet d'aller plus loin dans notre analyse, en nous appuyant sur un domaine concret, tout en gardant à l'esprit l'objectif de notre travail, qui consiste à modéliser les explications dans les SBC-RPC en couvrant au maximum l'étendue de la discipline. L'analyse KADS, comme nous l'avons décrite dans le chapitre III, repose sur la modélisation des connaissances. Celle-ci en fait une approche intéressante pour la génération d'explications qui repose avant tout sur une modélisation explicite des connaissances [Bourcier *et al.* 94]. Ce chapitre concerne la modélisation de la résolution de problème, la modélisation des explications qui repose sur le modèle d'expertise sera faite au chapitre VI. Néanmoins, celui-ci reposant sur les résultats de la modélisation du processus de résolution, c'est pourquoi les explications sont prises en compte lors de l'élaboration de chaque modèle de la résolution de problème. Nous allons donc commencer par présenter le contexte dans lequel nous nous plaçons dans le modèle de l'organisation. Ensuite nous présenterons les modèles de tâches, d'agents et de communication. Pour terminer nous décrirons le modèle d'expertise de la résolution de

problème. Nous verrons que l'instanciation des critères du modèle d'explications est réalisée lors de l'élaboration des différents modèles.

V.1 Le modèle d'organisation

Le modèle d'organisation de KADS a pour objectif de décrire l'environnement organisationnel de l'application qui nous intéresse et qui concerne le jardinage en carrés.

L'application que nous avons choisie comme support pour modéliser nos travaux sur la génération d'explications est le jardinage en carrés. Celle-ci n'a pas de contexte organisationnel défini. Cependant, nous pouvons facilement en imaginer un. En effet, plaçons-nous dans le contexte d'une entreprise de jardinage. On pourrait alors imaginer que celle-ci, en plus du service traditionnel de vente de plants potagers ou de fleurs, puisse offrir un service de conseil en jardinage. Celui-ci aurait pour rôle tout d'abord d'apprendre à sa clientèle les caractéristiques associées à chacune des plantes dont celle-ci fait l'acquisition, et ensuite de lui fournir, à partir de la liste des plants que les acheteurs veulent intégrer dans leur jardin, une proposition de disposition de ceux-ci. En effet, on peut penser que l'entreprise possède des spécialistes en jardinage qui ont déjà eu, par le passé, l'occasion de concevoir des plans de jardin pour les clients qui en auraient fait la demande. Elle aurait alors pu conserver ces plans pour une réutilisation future. Aussi, avec l'ensemble des plans déjà à sa disposition et de ceux qu'elle produit à la demande, et avec l'aide de personnes compétentes, la réalisation d'un système informatique facilitant ce service de planification présenterait un intérêt certain. En effet, il aiderait à former les nouveaux employés qui ne seraient pas familiers avec le domaine du jardinage. Ceux-ci pourraient apprendre seuls tout ce qu'il faut savoir sur les légumes et leur propriétés potagères, et comment composer un jardin pour le rendre le plus productif possible. De plus, l'utilisation du système qui permettrait de réaliser cette configuration apporterait un gain de temps considérable aux employés qui n'auraient alors pas à créer systématiquement une nouvelle composition mais juste à la faire trouver par le système. Une autre possibilité consisterait aussi à offrir ce service directement aux clients, en mettant le système à leur disposition. Ils composeraient alors leur jardin comme ils l'entendent en se fiant au système pour ne pas faire d'erreur de disposition et ainsi obtenir un rendement intéressant. Cependant, si les spécialistes en jardinage sont formés pour connaître les règles à respecter et les contraintes d'organisation

d'un plan selon les propriétés des plantes potagères, il n'en est pas de même pour les clients qui utiliseraient ce service ou pour les nouveaux employés n'ayant pas reçu de formation. C'est pourquoi, doter le système d'un dispositif lui permettant d'expliquer les plans générés et comment les élaborer représente le problème principal de l'application. Celle-ci offre donc l'ensemble des qualités requises pour justifier la conception d'un système. Le choix d'un SBC est déterminé par les critères que nous venons d'énoncer. En effet, les spécialistes ou experts existent, même s'ils peuvent être dispersés géographiquement, de même que l'expertise, déjà sous forme de plans, qui a été réalisée par le passé. De plus, le fait de réutiliser ces plans pour en élaborer de nouveaux fait du RPC le raisonnement approprié pour résoudre ce type de problème, car il permet d'utiliser les connaissances déjà existantes et de profiter de l'expérience et des succès ou échecs antérieurs pour offrir aux clients des configurations de jardin d'une grande fiabilité. Finalement, le fait de rendre le système accessible à ses futurs utilisateurs entraîne l'étude des explications qui devront être adaptées à la fois aux utilisateurs et au système même qui est le SBC-RPC.

Nous pouvons donc déterminer un contexte organisationnel dans lequel notre application pourrait être introduite. Cependant, celui-ci n'existant pas réellement, nous n'avons pas pu réaliser l'acquisition des connaissances en nous fiant sur l'expérience d'éventuels experts. Le choix du jardinage en carrés a été déterminé par l'accessibilité relativement facile du domaine. En effet, de nombreuses personnes font chez elles un jardin en y semant un assortiment de plantes potagères. Elles les disposent selon leur convenance, et, à partir des résultats plus ou moins satisfaisants qu'elles obtiennent, apprennent à connaître les conditions à respecter pour pouvoir à la fin de la saison obtenir une bonne récolte. N'ayant pas de jardin à disposition, nous avons usé de moyens divers pour constituer notre propre base de connaissances. Tout d'abord nous avons fait appel aux quelques notions personnelles que nous possédions sur les légumes. Ensuite, nous nous sommes basée sur deux ouvrages de références [Bartholomew 84; Gagnon 84] traitant du jardinage en carrés et des plantes potagères. Finalement nous avons fait appel à un dictionnaire pour compléter et trouver l'ensemble des définitions manquantes. Nous avons procédé à ce recueillement d'informations en gardant constamment à l'esprit le problème principal : pouvoir fournir à tout moment et en toutes circonstances des explications sur ce que nous traitions. Ceci

explique l'utilisation d'un dictionnaire qui reflète bien une forme d'explication des connaissances, aussi il représente un support idéal pour en créer.

Nous avons expliqué comment nous sommes devenue experte en légumes potagers et en jardinage en carrés, aussi il est nécessaire de définir chacune de ces notions. Ainsi, tout le monde sait ce qu'est un légume pour en avoir mangé, mais cela ne signifie pas que l'on connaisse par exemple leur famille d'appartenance. En effet, les légumes sont caractérisés par de nombreuses propriétés qui permettent de les distinguer. Ainsi, on reconnaît en général un légume à sa forme, à sa couleur et à son goût. Mais un légume ne se résume pas à ces critères-là. En effet, on peut étudier ceux-ci d'un point de vue biologique, ou même médical pour leurs vertus médicinales, on se servira alors davantage de ce qui caractérise la famille à laquelle ils appartiennent que de leur texture. D'un point de vue plus terre à terre, les légumes offrent également la particularité d'avoir entre eux des relations de compagnonnage. Ainsi, on peut définir si deux légumes sont amis, ennemis ou encore si l'un protège l'autre. Connaître cette relation est essentielle si l'on veut réaliser un jardin qui, le moment venu, produira l'ensemble des légumes que nous y avons plantés. Ainsi, le succès d'un jardin reposera sur le choix des légumes. Les jardins qui nous intéressent ici sont les jardins en carrés [Bartholomew 84]. Cette pratique de jardinage est avantageuse pour les personnes ne disposant pas de beaucoup d'espace et voulant récolter leurs propres légumes. Le principe est simple. Un jardin consiste en un plan de terre de 1m20 sur 1m20, chacun est constitué de 16 carrés plus petits de dimensions égales, soit 30 cm sur 30 cm. Dans chaque carré, on plante un type de légume. Il faut alors prendre en compte les propriétés des différentes sortes de légumes ainsi que les contraintes telles que le soleil, la nature de la terre, l'arrosage, la rotation des plantes d'une année sur l'autre, etc. pour organiser le jardin. Ainsi dans un jardin de 1m20 sur 1m20 on peut, par exemple, obtenir la récolte suivante [Bartholomew 84]:

- 32 carottes
- 2.5 kg de pois
- 16 radis
- 1 pomme de chou-fleur
- 9 navets japonais
- 18 bottes d'épinards
- 4 pommes de laitue romaine
- 16 betteraves
- 12 bottes de laitue à couper
- 1 pomme de chou
- 16 échalotes
- 1 pomme de brocoli

Nous avons, dans cette partie, montré quel pourrait être le contexte organisationnel de notre application, la méthodologie que nous avons suivie pour recueillir les connaissances du domaine faute d'experts en la matière à notre disposition, et finalement le domaine d'application lui-même, à savoir les légumes potagers et le jardinage en carrés. Ceci nous a

permis de réaliser l'étude de faisabilité de notre application en tenant compte que celle-ci ne s'inscrit pas dans un projet réel et donc à identifier certaines des contraintes dont nous devrons dans le futur tenir compte pour poursuivre la modélisation. Nous devons maintenant définir le problème de l'application de manière plus détaillée.

Comme nous l'avons déjà mentionné, l'application considérée concerne les légumes potagers et le jardinage en carrés. Pour identifier l'ensemble des fonctions qui devraient être supportées par celle-ci, nous avons dû nous mettre à la place de simples apprentis jardiniers et considérer quels seraient nos besoins si l'on avait un système à notre disposition spécialisé dans le jardinage. Nous essayons ici d'identifier les fonctions futures de notre système EDEN. Dans une analyse KADS contenant un contexte organisationnel bien défini avec une entreprise par exemple, l'étude veut que l'on analyse les fonctions de l'organisation avant l'introduction d'un SBC, ainsi que la structure de l'organisation et son processus. Nous ne disposons pas de ces éléments, c'est pourquoi nous nous limitons à l'étude des fonctions, des besoins et du problème de l'application de jardinage en carrés. Ainsi, l'application devrait remplir plusieurs fonctions indépendantes dans leurs rôles, mais dépendantes dans leurs connaissances. En effet, nous avons déjà défini que pour faire un jardin en carrés il est nécessaire de connaître certaines propriétés sur les légumes. Ceci permet donc d'établir un ensemble de contraintes à respecter. C'est pourquoi la première fonction repose sur la classification des légumes. Cette fonction présente plusieurs avantages. Avant tout, elle permet d'apporter un outil éducatif pour apprendre ce que sont les légumes. De plus elle peut même être considérée comme ludique afin de familiariser les enfants avec les plantes potagères et pourquoi pas leur apprendre à aimer les légumes. Enfin, elle sert de point de départ pour apprendre à composer un jardin en carrés productifs. Aussi, la deuxième fonction consiste à construire des plans de jardin en carrés qui soient cohérents, c'est-à-dire qui produisent un bon rendement. Cette fonction peut être réalisée comme nous l'avons mentionné dans le contexte organisationnel soit comme outil d'apprentissage, mais également pour simplement informer les clients d'un service qu'on leur offre pour maximiser leur contentement et ainsi les inciter à revenir. Cependant, les plans élaborés pouvant ne pas être à la satisfaction complète des clients, la troisième fonction, qui représente en fait le problème que nous nous proposons de résoudre ici, consisterait donc à produire des explications permettant d'apporter des informations supplémentaires sur les difficultés

rencontrées par les clients ou les utilisateurs du système à l'encontre des fonctions de classification et de planification. Pour identifier la nature de ce problème nous devons reprendre les fonctions précédemment définies c'est-à-dire la classification et la planification et déterminer les sources et la nature des difficultés. Ainsi, dans un contexte réel, tout d'abord le vendeur qui offrirait le service de classification de légumes donnerait les éléments de base nécessaires pour comprendre en quoi consiste la classification d'un légume potager. Ensuite, il pourrait aider le client à voir la pertinence de la famille identifiée, à comprendre comment celle-ci a été retrouvée, sur quelles valeurs la recherche a été effectuée. Ensuite, un vendeur expliquerait au client, soit spontanément soit à sa demande, comment il compose le jardin. Il pourrait aussi le renseigner sur la méthode qu'il utilise pour configurer le jardin, sur les contraintes à respecter ou encore l'informer sur les propriétés de certains légumes. C'est pourquoi, il est nécessaire de reproduire ce comportement explicatif, inné chez la personne, à l'intérieur du système pour que celui-ci soit accessible et fournisse les mêmes services que s'il s'agissait d'un véritable vendeur. Ainsi, le client qui va se voir offrir un nouveau plan par le système va vouloir comparer les différents plans entre eux, émettre des préférences. Il faut alors le persuader de la pertinence du plan retrouvé et justifier le choix qui a été fait. Si de plus, on considère que le client peut arriver avec son propre plan de jardin déjà constitué, il va s'adresser au vendeur dans le but de le faire vérifier et valider pour s'assurer qu'il est correct mais aussi pour le faire analyser afin de détecter les erreurs de disposition, dans le cas où il l'aurait essayé et qu'il n'aurait pas obtenu le rendement attendu. Nous venons donc d'établir les besoins en explications que rencontre l'application de RPC que nous traitons.

Comme nous pouvons le constater, l'élaboration du modèle d'organisation tel que nous en avons donné la définition dans le chapitre III n'est pas possible dans notre cas. En effet, nous avons montré que le contexte organisationnel, s'il peut être facilement établi, n'est pas réel. C'est pourquoi, nous ne pouvons procéder à l'analyse de la structure générale et du processus de l'organisation. De même nous pouvons imaginer quelles fonctions existeraient. Les seules fonctions que nous avons identifiées sont celles de notre application, en considérant qu'elles pourraient être réellement exécutées par des employés dans un contexte réel. Nous avons néanmoins pu définir l'ensemble des agents qui interviennent dans l'application, soit les vendeurs expérimentés ou non formés, les clients enfants ou adultes et le système même, ainsi que les contraintes dont il faut tenir compte qui sont liées au domaine d'application, soit

les légumes potagers et le jardinage en carrés avec les règles de compagnonnage. Toutefois, si le modèle d'organisation n'est pas forcément adapté pour une application d'études, il s'avère utile pour procéder à l'instanciation de certains des critères du modèle générique d'explications. Tout d'abord, il permet d'identifier à qui sont destinées les explications et à quel moment elles doivent être produites. De plus, nous avons, en définissant le contexte, montré quels buts devait atteindre le système pour résoudre les problèmes liés à la classification et à la planification réalisées par l'application. Parmi ces buts nous retenons les fonctions d'information, de familiarisation, d'apprentissage, de jeu, de conseil, de vérification, de validation, de justification, de persuasion et de vente. L'identification de ce but sera utile lorsqu'une explication sera requise car il permet de répondre aux questions "Pourquoi expliquer ?" et "Quoi expliquer ?" qui sont prises en compte lors de l'élaboration du modèle de tâches que nous décrivons dans le paragraphe suivant.

V.2 Le modèle de tâches

Il consiste à identifier, dans l'environnement organisationnel, une décomposition hiérarchique des tâches à réaliser. Selon la méthode KADS, il se décompose en quatre étapes dont la première est de décomposer le problème général en fonction des éléments organisationnels.

V.2.1 Décomposition du problème général en fonction des éléments organisationnels

L'application de jardinage, comme nous l'avons mentionné, nécessite la résolution d'au moins deux types de problème, elle fait donc appel à plusieurs types de tâches. Tout d'abord elle consiste en une tâche de classification. En effet, elle permet à un utilisateur de pouvoir apprendre la classification des légumes selon leur famille d'appartenance. L'utilisateur donne la description du légume dont il veut connaître la famille d'appartenance conformément aux descripteurs qui lui sont demandés et en fonction de ses connaissances. Ensuite le processus de résolution indique à quelle famille le légume appartient. Plusieurs schémas de situation peuvent se présenter. Si l'utilisateur connaît le nom du légume, le système n'aura qu'à rechercher dans sa base de connaissances le légume en question et fournir la réponse correspondante, soit la classe associée retrouvée. Un autre schéma peut se présenter dans lequel l'utilisateur ne connaît pas le nom du légume. Le système, d'après la description faite

par l'utilisateur, devra alors retrouver la famille du légume par un processus qui lui permettra de retrouver les classes de légumes qui semblent être les plus pertinentes par rapport aux données du problème posé. Le deuxième type de résolution consiste en une tâche de configuration de jardin en carrés. L'utilisateur énumère la liste des légumes qu'il veut planter dans son jardin et le système retrouve les plants qui sont le plus en accord avec ses exigences. En fait on peut considérer le résultat comme un exemple que l'utilisateur est libre d'accepter ou de refuser en fonction de son opinion. Ainsi, le système, avant de fournir un résultat, doit vérifier la cohérence de celui-ci. Si la configuration est validée, elle est fournie à l'utilisateur. Si le système décèle des erreurs dans la configuration, il la présente à l'utilisateur en indiquant à ce dernier les erreurs qu'elle comporte. Ensuite, l'utilisateur peut apporter des modifications au contenu du plan en fonction de ses besoins et de ses préférences. Le système devra ensuite vérifier et valider la solution trouvée par l'utilisateur. Si aucune solution n'est trouvée, un processus de configuration doit être élaboré pour que le système crée lui-même ses propres plans. Le problème associé à ces deux tâches de résolution est qu'elles font toutes deux appel à des connaissances et des méthodes, pour retrouver des familles de légumes ou des jardins déjà configurés en accord avec les descriptions fournies, qui ne sont pas forcément familières ou compréhensibles aux utilisateurs. C'est pourquoi il s'avère indispensable de produire un dispositif explicatif pour rendre l'application plus accessible. De plus, étant donné que le raisonnement employé consiste à réutiliser de la connaissance antérieure pour trouver des solutions, la tâche explicative consiste également à faire comprendre ce phénomène que nous avons introduit plus tôt comme le RPC.

V.2.2 Le modèle de tâches initial

Le modèle de tâches initial spécifie comment la fonction est décomposée par l'expert en tâches et sous-tâches, indique leurs entrées-sorties et les agents internes ou externes qui les exécuteront. L'élaboration du modèle de tâches a été motivée par les objectifs suivants :

- connaître les domaines des légumes potagers et du jardinage en carrés;
- apprendre à classifier une légume selon sa famille d'appartenance;
- apprendre comment planifier un jardin;
- déterminer ce qui va poser problème lors de la réalisation des objectifs précédents.

La tâche principale de notre application en est une d'explication. Elle se décompose en deux tâches, expliquer le domaine d'application d'une part et expliquer le RPC d'autre part. La deuxième se décompose également en deux tâches : expliquer la tâche de classification et expliquer la tâche de planification. Finalement, pour chacune d'elle, il faut considérer les tâches mêmes de classification et de planification qu'il est indispensable de connaître si l'on veut fournir des explications compréhensibles qui donnent aux utilisateurs la meilleure satisfaction possible. C'est ce que nous considérons comme étant la deuxième tâche, à savoir la tâche du RPC. La figure 7 présente le modèle des tâches initial tel que nous venons d'en faire la décomposition.

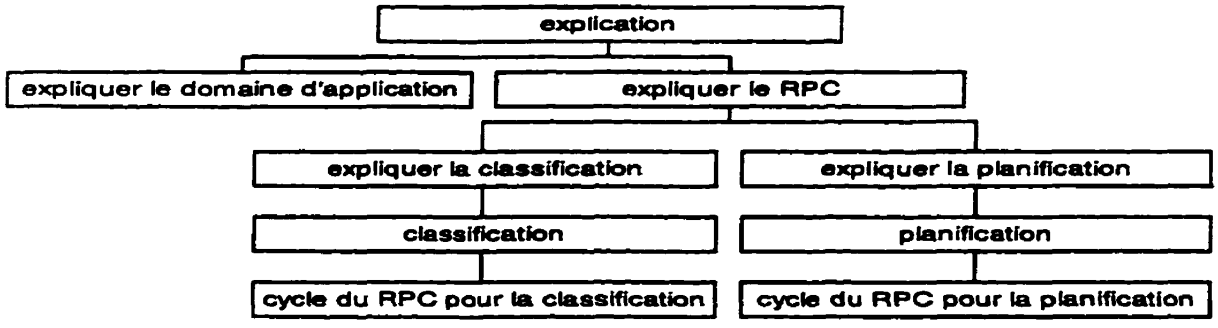

Figure 7. Modèle de tâches initial

Nous allons dans ce qui suit donner le schéma général de chacune des tâches en fonction des paramètres donnés dans KADS. Nous ne donnons pas la décomposition complète des tâches, celle-ci sera décrite lors de l'analyse du niveau des tâches du modèle d'expertise.

- Tâche **explication :**

  Agent : système

  Informations en entrée : énoncé de la requête

  Informations en sortie : explication

  But à atteindre : identifier l'objet de la question, répondre à la question posée en accord avec son contenu et les caractéristiques de l'agent qui en est à l'origine, donc du modèle de l'utilisateur, buts de l'explication et des raisons de l'explication.

  Fréquence : à chaque utilisation du système, n'importe quand au cours de la résolution

  Sous-tâches :

  expliquer la connaissance du domaine

  expliquer le RPC

- Tâche **expliquer la connaissance du domaine :**

  Agent : système

  Informations en entrée : données

  Information en sortie : définition, exemples, descriptions, etc.

  But à atteindre : fournir une explication appropriée

  Sous-tâches :

  Identifier le type d'explication (décrite dans le chapitre VI)

  Élaborer ce type d'explication (décrite dans le chapitre VI)

- Tâche **expliquer le RPC :**

  Agent : système

  Informations en entrée : un résultat, un cas, toute donnée relative au RPC

  Informations en sortie : une explication

  But à atteindre : identifier le type de tâche RPC sur laquelle porte la demande d'explication et procéder à son élaboration.

  Sous-tâches :

  Expliquer la tâche de classification (décrite dans le chapitre VI)

  Expliquer la tâche de planification (décrite dans le chapitre VI)

- Tâche de **classification** :

  Agent : système

  Informations en entrée : données d'un légume

  Information en sortie : famille du légume

  But à atteindre : trouver avec le maximum possible de certitude une classe d'appartenance au légume donné en entrée

  Sous-tâches : RPC pour la classification

- Tâche de **planification :**

  Agent : système

  Informations en entrée : liste de légumes

  Informations en sortie : plan d'un jardin

  But à atteindre : configurer un jardin en respectant les contraintes de compagnonnage du jardinage, en lui assurant un bon rendement et en accord avec les données d'entrée

  Sous-tâches: RPC pour la planification :

  Retrouver un ancien plan

Créer un nouveau plan

Adapter un plan

Vérifier et valider un plan

- Tâche **RPC :**

  Agent : système

  Informations en entrée : données d'un cas

  Informations en sortie : cas solution

  But à atteindre : retrouver le cas qui soit le plus similaire au cas donné en problème afin de récupérer sa solution pour l'adapter au mieux et résoudre le problème posé.

  Sous-tâches :les sous-tâches sont effectuées en fonction du type de la tâche classification ou planification

  Retrouver

  Réutiliser

  Réviser

  Retenir

On trouve la décomposition en tâches et en sous-tâches du RPC dans la figure 8.

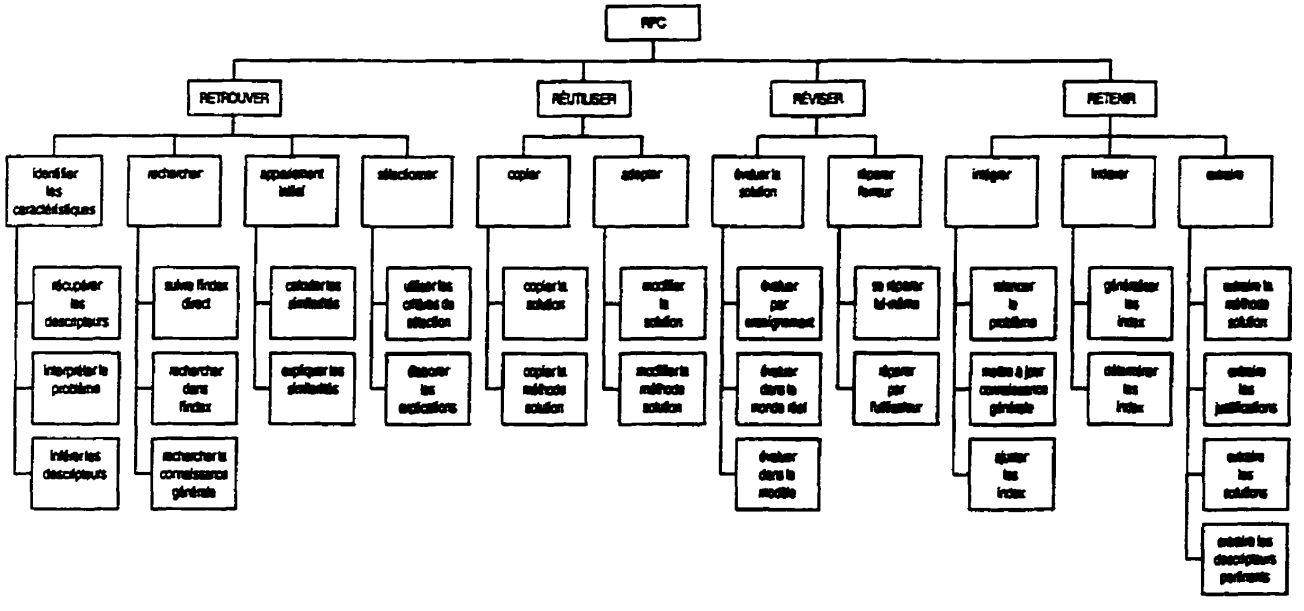

Figure 8. Décomposition en méthodes et tâches du RPC [Aamodt&Plaza 94]

Le modèle de tâches comprend donc la décomposition en tâches et en sous-tâches du raisonnement suivi par l'expert pour résoudre les problèmes. Nous avons donc déterminé ici les principales tâches que nous allons devoir analyser et traiter selon la méthode KADS. Le

modèle de tâches comprend l'élaboration de deux autres modèles : le modèle de tâches nécessaires et le modèle de vérification. L'une comme l'autre réfèrent aux contraintes d'assignation des tâches aux agents de l'organisation et du système. Dans notre cas, toutes les tâches sont réalisées par le système mis à part la tâche d'énoncer le problème qui dépend de l'utilisateur. En effet, c'est lui qui fournit les informations en entrée, que celles-ci concernent la description d'un légume, la liste de légumes à utiliser pour construire un jardin ou encore le sujet d'une demande d'explication. Nous allons dans la section suivante décrire le modèle d'agents.

V.3 Le modèle d'agents

Ce modèle consiste à allouer à des agents spécifiques les tâches définies dans le modèle de tâches en fonction de leur compétences, ainsi qu'à définir chacun des agents intervenant dans l'application.

Nous avons identifié deux agents principaux dans notre application : l'agent utilisateur et l'agent système (figure 9). L'agent utilisateur peut être répertorié selon quatre catégories. La première concerne l'utilisateur novice, c'est-à-dire celui qui utilise le système comme outil d'apprentissage car il n'est pas familier avec le domaine, soit le client qui peut être un enfant ou un adulte, ou un vendeur qui doit recevoir une formation dans le domaine du jardinage afin de devenir expert. La deuxième catégorie regroupe les utilisateurs qui sont déjà experts avec le jardinage et qui utilisent le système comme outil pour conseiller les clients et gagner du temps. La troisième catégorie s'adresse aux utilisateurs qui conçoivent le système et qui veulent vérifier que celui-ci fonctionne selon les conditions établies. Une dernière catégorie indépendante du domaine d'application doit être considérée. Elle concerne les utilisateurs qui désirent apprendre le mode de fonctionnement du RPC, les techniques de classification et de planification. Pour chacun d'eux, nous devons définir un modèle qui sera utilisé lors de la génération d'explications afin d'adapter celles-ci à leur niveau de compétence et à leurs exigences et besoins. Cependant, étant donné que notre application ne se situe pas dans un contexte réel avec des utilisateurs dont la description précise est définie, nous considérons que le modèle de l'utilisateur est tel que nous venons de le décrire, à savoir chaque catégorie d'agent avec pour chacun d'eux leur niveau de compétence. Ainsi, si l'utilisateur est un enfant nous allons favoriser une représentation des explications sous une forme conviviale avec

l'utilisation de graphiques par exemple. Aussi, c'est le contenu du modèle tel que nous l'avons défini qui va nous servir de support pour faire les choix d'interfaces et de modes de communication entre les utilisateurs et le système. Celui-ci représente le deuxième agent. Il peut également être décomposé en plusieurs agents. Ainsi, le premier est l'agent des connaissances, c'est lui qui détient toute la connaissance de l'application. Là aussi, il faut distinguer l'agent des connaissances d'application qui contient la connaissance relative aux processus de classification et de planification de l'agent des connaissances d'explications qui utilise des informations qui ne sont pas nécessaires pour accomplir les tâches de planification et de classification. Ensuite nous avons un agent pour la résolution de problème qui se décompose en deux agents, un pour la tâche de classification et un pour la tâche de planification. Finalement, le dernier agent à considérer est l'agent d'explication. Celui-ci a pour rôle de générer des explications relatives à l'ensemble des tâches effectuées par les autres agents. La communication entre les divers agents et les contraintes associées sont représentées dans le modèle suivant qui est le modèle de communication.

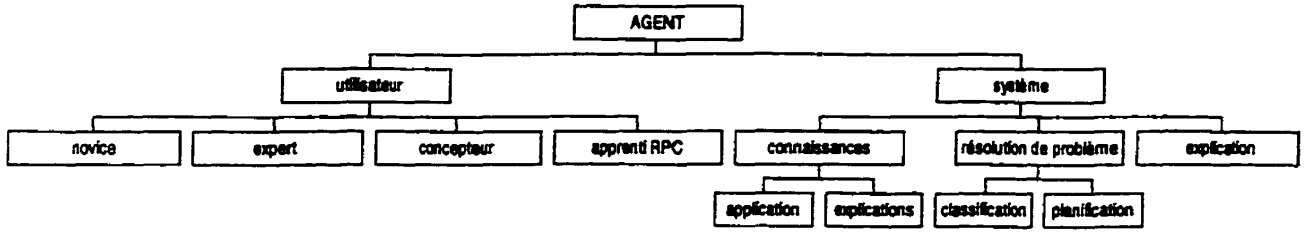

Figure 9. Modèle des agents

V.4 Le modèle de communication

Il doit spécifier l'environnement communicatif dans lequel le SBC doit fonctionner en décrivant les échanges d'information entre les agents qui exécutent la même tâche, en fournissant les concepts et les mécanismes de communication en fonction des conditions établies et des compétences des agents communicants.

Le modèle de communication permet donc d'établir la nature du dialogue entre le système et les utilisateurs. Il représente la base pour les décisions d'interfaçage qui seront prises lors de la phase de conception. De plus, il permet de modéliser la coopération et la communication entre les agents qui évoluent dans le système même. Pour élaborer le modèle de communication il est nécessaire de connaître le modèle de tâches défini en section V.2. La

communication entre les agents du système a été décrite dans les modèles de tâches et d'agents. En effet, elle consiste à assigner aux agents les tâches à réaliser. C'est pourquoi, n'ayant pas encore réalisé de manière précise l'analyse explicative, nous reviendrons sur l'élaboration du modèle de communication au cours de l'élaboration du modèle d'expertise (Section V.5 et chapitre VI). Cela nous permettra d'être plus explicite.

V.5 Le modèle d'expertise

Le but de ce modèle consiste à spécifier l'expertise de résolution de problèmes en utilisant le modèle de tâches et en décrivant le comportement et le type des connaissances nécessaires au système pour la résolution de problème, et ce indépendamment de toute contrainte d'implémentation. On distingue quatre niveaux d'abstraction dans le modèle d'expertise tels qu'ils ont été décrits en III.2.1.1. Nous allons dans ce qui suit présenter l'approche que nous avons suivie pour procéder à cette modélisation.

V.5.1 Description de l'approche utilisée

Comme nous l'avons décrit dans le modèle de tâches, l'application de jardinage a pour objectif de résoudre trois types de problèmes. Les deux premiers concernent le RPC pour les tâches de planification et de classification. Celles-ci sont des tâches spécifiques aux SBC en général et sont particulièrement adaptées au RPC pour les raisons que nous avons évoquées en II.2.3. La troisième tâche, qui est la plus importante dans le cadre de nos travaux de maîtrise, est la tâche d'explication. En fait, les deux premières sont indispensables et doivent être étudiées afin de pouvoir dresser le contexte de modélisation des explications dans les SBC-PRC. L'approche suivie est inspirée de celles de [Martin 94; Springer 93, Baumewerd-Ahlmann *et al.* 90; Bourcier *et al.* 94] et adaptée aux SBC-RPC. En effet, ces travaux ne portent pas sur le RPC mais sur des systèmes utilisant un raisonnement à base de règles d'inférences. De plus, leur objectif est de produire des explications en langage naturel ce qui n'est pas notre cas. En effet, notre objectif est de générer un ensemble représentatif d'explications qui soient propres au RPC. Aussi, comme Bourcier, nous pensons que la génération d'explications étant une tâche à part entière [Swartout&Moore 93], et il est donc nécessaire de procéder à une analyse spécifique de celle-ci. C'est pourquoi, nous considérons la nécessité de produire un modèle d'expertise spécifique à la tâche d'explications. À la différence de Martin [Martin 94] qui élabore aussi un modèle d'expertise propre aux explications, nous ne créons pas de modèles d'expertise pour la communication et pour la coopération, les connaissances

explicatives que nous avons recueillies pour chacun des modèles étant suffisantes pour nos travaux. Cependant, comme nous l'avons constaté plus tôt, les connaissances utiles à la génération d'explications consistent à la fois en des connaissances propres à la résolution de problème, c'est-à-dire des connaissances utilisées par les tâches de classification et de planification et donc contenues dans leurs bases de connaissance respectives, mais aussi en des connaissances qui sont indépendantes du processus de résolution mais indispensables à la bonne compréhension des utilisateurs. Nous réalisons donc 3 modèles d'expertise, un pour chacune des tâches énoncées, planification et classification qui forment le modèle d'expertise du RPC, ensuite on crée un modèle d'expertise spécifique aux explications. Ce dernier sera décrit dans le chapitre suivant.

V.5.2 Élaboration du modèle d'expertise pour le RPC

L'application de jardinage permettant de résoudre deux type de tâches du RPC, c'est pourquoi nous détaillons pour chacune d'elles le modèle d'expertise associé. Certains concepts et relations sont communs aux deux modèles. Le premier modèle est celui de la classification.

V.5.2.1 Modèle d'expertise pour la classification

a) Le niveau du domaine

Ce niveau permet de décrire la connaissance statique liée à la tâche de classification indépendamment de son implémentation. Elle concerne uniquement les informations relatives aux légumes et ce qui permet de les classifier. Ainsi, chaque légume est décrit sous forme d'un cas qui correspond à l'instanciation d'un cas type ***légume*** en fonction d'un ensemble de descripteurs définis et d'une solution. La description d'un cas type ***légume*** est la suivante.

Cas légume :
Le problème:

- **n° d'identification :**

numéro du cas

- **nom :**

carotte, chou, navet, ...

- **forme végétative :**

bouquet, bourgeon, bulbes, côtes, feuilles, fleur, fruit, gousse, graine, influorescence, pied, pousse, racine, tige, tubercule;

- **variété :**

a des ressemblances avec d'autres légumes tels que l'ail, la laitue, le chou, etc. exemple : la batavia est de la variété de la laitue;

- **couleur :**

argenté, rose, rouge, orange, vert, violet, jaune, noir, marron, bicolore;

- **cuisine :**

condiment, salade, légume, crudité, fruit;

- **plante :**

potagère, herbacée, aromatique, sauvage;

- **vie :**

annuelle, bisannuelle, vivace.

La solution :

- **famille :**

nom de la famille à laquelle appartient le légume:

chénopodiacées, composées, crassulacées, crucifères, cucurbitacées, labiacées, légumineuses, liliacées, ombelliféracées, papilionacées, solanacées, valérianacées, inconnue.

Exemple de cas:

Cas n° 7:

- *Nom* aubergine
- *Forme végétative* fruit
- *Cuisine* légume
- *Vie* annuelle
- *Variété* aubergine
- *Couleur* violette
- *Plante* potagère
- *Famille:* cucurbitacées

Cas n° 9 :

- *Nom* batavia
- *Forme végétative* feuille
- *Cuisine* salade
- *Vie* annuelle
- *Variété* laitue feuille
- *Couleur* verte
- *Plante* potagère
- *Famille* composées

Nous venons donc de décrire les seules connaissances nécessaires à la tâche de classification de légumes potagers. La similarité entre deux légumes sera déterminée en fonction des descripteurs que nous avons définis. Les autres connaissances sont des connaissances liées à la résolution de problème. Aussi elles n'ont pas à être décrites dans cette section. En effet, la tâche de classification consiste uniquement à prédire la valeur d'une propriété de l'ensemble des attributs décrivant un objet.

b) Le niveau des inférences

À partir des structures génériques d'inférences de la bibliothèque générique, nous avons réalisé nos propres structures d'inférences pour les adapter au problème en fonction de nos besoins. La tâche de classification que nous traitons est particulière. En effet, c'est une tâche de RPC. Cependant, nous avons défini dans les modèles précédents un certain nombre de contraintes la concernant. Aussi nous avons déterminé que seule la phase RETROUVER du RPC devait être appliquée. Les processus suivants du cycle du RPC sont effectués par l'utilisateur. Il n'y a pas de phase d'adaptation automatique pour la classification, comme nous l'avons mentionné dans le modèle d'organisation (section V.1), l'adaptation étant faite manuellement par l'utilisateur.

Nous avons donc identifié quatre structures d'inférence en considérant également les tâches réalisées par l'utilisateur pour l'adaptation, la validation et l'enregistrement. Nous donnons la description des structures d'inférences. À l'instar de Martin [Martin 94], le formalisme utilisé dans ce mémoire pour représenter les structures d'inférence est le suivant :

- Sources de connaissances (SC) : représentées graphiquement par une ellipse et textuellement en caractères gras. Elles sont représentées textuellement par Martin [Martin 94] par le terme Structures de Tâches (ST). En effet, les SC présentées ici sont soit élémentaires, c'est-à-dire qu'on ne peut aller plus loin dans leur décomposition, soit non-élémentaires, elles font alors appel à d'autres SC, d'où leur appellation de ST. Nous les nommerons ici pareillement SC.
- Métaclasses (MC) : représentées graphiquement par une boîte et textuellement en italique.

- **Structure d'inférence de la tâche RECHERCHER** (figure 10): elle permet d'analyser les données du problème et de les comparer afin de retrouver un cas qui soit le plus similaire.

Les structures de tâches associées sont représentées selon le modèle suivant :

SC **nom_de_la_SC**
(Entrée *nom_des_MC* en entrées;
Sortie *nom_des_MC* en sortie)
Méthodes : Types de méthodes utilisées pour réaliser la tâche;
Connaissances : Types de connaissances utilisées.

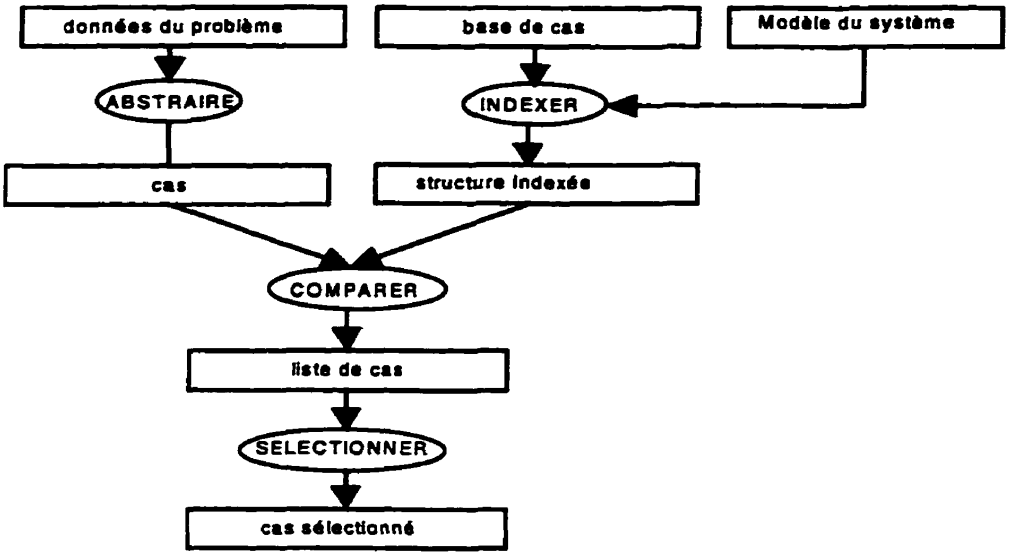

Figure 10 SI de la tâche *Rechercher*

SC **abstraire**

(Entrée *données du problème* : structure contenant les données observées du problème

Sortie *cas* : structure contenant les données correspondant aux descripteurs du cas)

Méthodes Instancier le modèle de cas avec les valeurs données.

Connaissances Signification des descripteurs

SC **indexer**

(Entrée *modèle-du-système* : contient les principes de classification de cas;

base-de-cas : base contenant tous les cas déjà résolus,

Sortie *structure-indexée* : arbre de décision construit)

Méthodes Technique d'induction; méthode de classification

Connaissances Induction.

SC **comparer**

(Entrée *structure-indexée, cas;*

Sortie *liste-de-cas* : avec les classes retrouvées suite au parcours de l'arbre)

Méthodes Recherche dans l'arbre de décision, exploration des branches, comparaison des données du cas avec les nœuds de l'arbre;

Connaissances Règles de comparaison, calcul des différences.

SC **sélectionner**

(Entrée *liste-de-cas* : liste des classes retrouvées avec leur score associé;

Sortie *cas-sélectionné* : classe qui est la plus satisfaisante conformément aux données du problème en fonction du résultat obtenu)

Méthodes techniques de sélection, explications des différences, justification de la pertinence,

aide à l'utilisateur dans son choix;

Connaissances connaissances de base du domaine, connaissances explicatives, techniques de sélection.

- **Structure d'inférence de la tâche ADAPTER** (figure 11) : elle permet de transformer le cas problème en fonction du cas retrouvé.

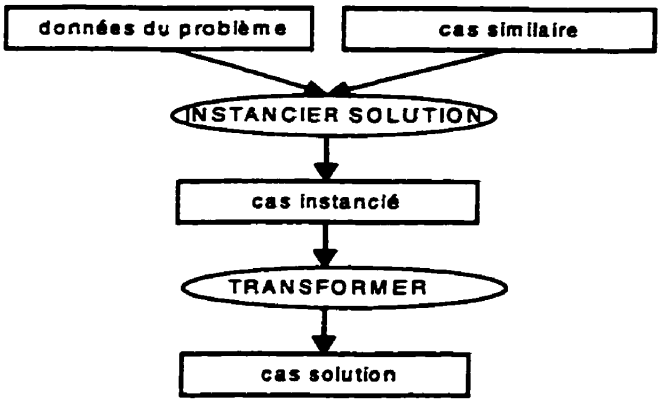

Figure 11 SI de la tâche *Adapter*

SC **instancier solution**

(Entrée *données-du-problème* : le cas tel qu'il est,
cas-similaire : cas qui peut être assorti au cas problème;

Sortie *cas-instancié* : cas auquel on a attribué les données du problème et la solution du cas similaire)

Méthodes Méthode d'appariement, d'instanciation;

Connaissances Règles de comparaison, d'instanciation de caractéristiques.

SC **transformer**

(Entrée *cas-instancié;*

Sortie *cas-solution* : cas qui contient les données du problème et la solution adaptées en fonction des contraintes d'adaptation)

Méthodes Techniques d'adaptation

Connaissances: Contraintes du domaine, règles d'adaptation, de transformation.

- **Structure d'inférence de la tâche VÉRIFIER et VALIDER** (figure 12) : elle permet de vérifier et de valider le cas transformé.

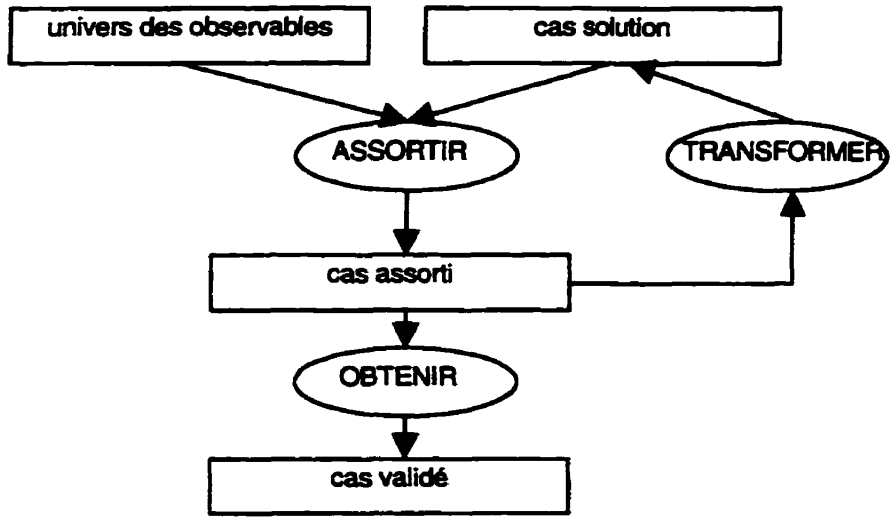

Figure 12 SI de la tâche *Vérifier* et *Valider*

SC **assortir**

(Entrée	*univers-des-observables* : tout les concepts, attributs–valeurs associés aux objets manipulés,
	cas-solution : un cas résolu avec les données d'un problème et la solution associée;
Sortie	*cas-assorti* : cas vérifié en accord avec l'univers des observables)
Méthodes	Méthodes de vérification
Connaissances	Contraintes du domaine

ST **transformer**

(Entrée	*cas-assorti*,
Sortie	*cas-solution* : cas transformé car il n'était pas valide)
Méthodes	Méthodes de transformation
Connaissances	Contraintes, règles de modification et de comparaison.

SC **obtenir**

(Entrée	*cas-assorti*
Sortie	*cas-validé* : cas représentant la solution finale du problème et validé)
Méthodes	Technique de validation
Connaissances	Règles de validation

- **Structure d'inférence de la tâche ENREGISTRER (figure 13) : elle permet d'intégrer à la base de cas le nouveau cas et de montrer la solution.**

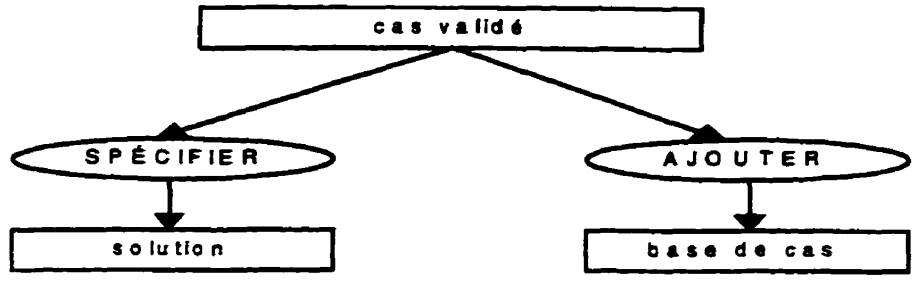

Figure 13. SI de la tâche *Enregistrer*

SC **spécifier**

(Entrée	*cas-validé;*
Sortie	*solution* : telle qu'elle est donnée)
Méthodes	Gestion interface système-utilisateur, produire la solution;
Connaissances	Formalisme de représentation de la solution, types d'échanges entre l'utilisateur et la machine.

SC **ajouter**

(Entrée	*cas-validé;*
Sortie	*base-de-cas*)
Méthodes	Insérer et ajouter un enregistrement dans une base;
Connaissances	Gestion d'une base de cas, les méthodes d'insertion, d'indexation.

c) Le niveau des tâches

Ce niveau considère la manière d'organiser les connaissances en fonction du but à atteindre. Il explicite comment contrôler les inférences élémentaires pour atteindre un but. Chaque tâche est décrite en utilisant le modèle suivant :

tâche *nom de la tâche*
but : but de la tâche
structure de tâche *nom de la tâche* { *description de la tâche* }

Le formalisme utilisé est inspiré de celui de Martin [Martin 94]. Ainsi la structure de tâche correspond à l'organisation des SC d'une SI en utilisant des facteurs d'itération tels que RÉPÉTER ... TANT QUE. Les sources de connaissances sont représentées sous la forme :

Nom de la SC(liste des MC en entrée (séparées par des virgules) -> MC en sortie)

La tâche de classification telle que nous l'avons définie se décompose donc en quatre tâches :

tâche rechercher

but : rechercher un cas dont les valeurs des descripteurs en fassent un cas similaire au cas problème

structure de tâche : rechercher (données du problème -> cas sélectionné)

{ abstraire (données-du-cas -> cas)

| indexer (base-de-cas, modèle-du-système -> structure indexée)

comparer (cas, structure-indexée -> liste-de-cas)

RÉPÉTER

sélectionner (liste-de-cas -> cas-sélectionné)

TANT QUE pertinence-cas-retrouvé = faux|

tâche adapter

but : utiliser la solution du cas retrouvé pour résoudre le cas courant, et adapter le cas ainsi trouvé afin qu'il réponde aux conditions de départ.

structure de tâche : adapter (données-du-problème, cas-similaire -> cas-solution)

{ instancier-solution(données-du-problème, cas-similaire -> cas-instancié)

RÉPÉTER

transformer(cas-> cas-solution)

TANT QUE transformation-à-faire = vrai}

tâche vérifier

but : obtenir un cas résolu dont la solution est correcte et dont le contenu est validé

termes de contrôle : assortiment-échoué

structure de tâche : vérifier (univers-des-observables, cas-solution->cas-validé)

{ assortir(univers-des-observables, cas-solution -> cas-assorti)

RÉPÉTER

transformer (cas-assorti -> cas-solution)

assortir(univers-des-observables, cas-solution -> cas-assorti)

TANT QUE assortiment-échoué = vrai

obtenir (cas-assorti -> cas-validé)}

tâche enregistrer

but : enrichir la base de cas en y insérant un nouveau cas résolu et fournir la solution finale à l'utilisateur

structure de tâche : enregistrer(cas-validé -> problème-terminé)

{ spécifier(enregistrement-validé -> solution) | ajouter(enregistrement-validé -> base-de-cas)}

d) Le niveau de stratégie

Le niveau de stratégie contrôle l'exécution de la tâche en fonction du modèle du système. C'est ici que l'on intègre les contrôles du type *SI condition ALORS action*, etc. De plus nous énonçons les types d'échanges réalisés entre le système et l'utilisateur à chaque étape.
Nous reprenons donc pour chacune des tâches que nous avons décrites les contrôles qui doivent être réalisés.

- **Rechercher** : l'utilisateur *identifie les caractéristiques* propres au légume. Si certaines informations lui font défaut, il peut en faire la demande au module d'explication. Enuite, à partir des *descripteurs pertinents* fournis par le système et l'arbre de décision créé à partir des données déjà disponibles et de l'algorithme d'induction choisi, le système fait une recherche. Ensuite, si des cas similaires ont été retrouvés en accord avec le taux de satisfaction préétabli, l'utilisateur *sélectionne* un cas parmi les cas les plus similaires retrouvé en fonction des règles de priorité de choix qu'il a déterminées et qu'il a obtenues soit par lui-même, soit avec l'aide d'explications. S'il n'est pas satisfait des cas retrouvés ou de la sélection obtenue il peut demander des explications sur comment les cas ont été retrouvés, et ce qui rend un cas plus satisfaisant qu'un autre ou enfin pour comprendre pourquoi un cas n'a pas été choisi.
- **Adapter** : si une solution est trouvée, alors elle est assignée au problème courant. L'*adaptation* est faite si aucun cas n'est retrouvé, l'expert attribue une solution au cas. Une *adaptation* doit aussi être faite si tous les descripteurs du problèmes n'ont pas été déterminés au départ ou nécessitent des modifications. Dans tous les cas, elle est laissée au soin de l'utilisateur et ne nécessite donc pas de traitement spécifique. Là aussi, si il n'est pas satisfait ou est confronté à des difficultés il peut réclamer de l'aide sur les descripteurs manquants ou sur les éléments qui lui permettraient de procéder seul à l'adaptation.
- **Vérifier** : une fois que l'expert a adapté le cas à la solution prise du cas sélectionné, il doit valider le cas obtenu. Si il y a des erreurs, l'expert les modifie en tenant compte des contraintes spécifiées, sinon il valide le cas.
- **Enregistrer** : quand les étapes d'adaptation et de vérification sont terminées, le système montre le cas final à l'utilisateur. Celui-ci peut alors choisir d'intégrer le nouveau cas et sa solution dans la base de cas.

Nous pouvons constater que pour la classification, seul le processus qui permet de retrouver un cas pour procéder à sa classification est réalisé par le système, les autres étapes sont exécutées par l'utilisateur et ne sont ni soumises à des contraintes, ni à des traitements propres au système. Elles ne sont pas obligatoires.

Nous venons donc de décrire le modèle d'expertise associé à la tâche de résolution de problème de classification. Nous pouvons déjà énoncer que certaines parties du modèle d'expertise de la planification seront similaires, les deux tâches de résolution suivant le même raisonnement, le RPC.

V.5.2.2 Modèle d'expertise pour la planification

La planification est la deuxième tâche de résolution de notre application. C'est une tâche de synthèse plus complexe que les tâches d'analyse classiques telles que la classification, comme nous l'avons mentionné dans le chapitre II.2.3, car elle nécessite une phase d'adaptation. Toutefois, elle présente quelques similitudes avec la classification qui seront présentées au moment opportun.

a) Niveau des connaissances

Le domaine des connaissances pour la planification de jardins en carrés est plus riche que celui de la classification de légumes. En effet, afin de configurer un jardin il est nécessaire de posséder des connaissances qui concernent à la fois les légumes et leurs caractéristiques mais également les règles et contraintes associées au jardinage.

Tout d'abord, un jardin en carrés ou plan est composé de 16 carrés de dimensions égales, soit 30cm sur 30cm. Sur chaque carré est semé un type de légume. Cependant les légumes, suivant qu'ils sont rampants, grimpants ou racines demandent plus ou moins d'espace pour pousser convenablement. C'est pourquoi il est nécessaire de connaître la surface nécessaire à chaque légume soit la longueur et la largeur minimales dont ils ont besoin. Dans le même ordre d'idée, certains légumes ont besoin de soleil pour pousser tandis que d'autres ont besoin d'ombre. Aussi, il faut connaître la hauteur que chaque plante peut atteindre et aussi l'orientation du plant par rapport au soleil afin d'éviter certaines situations problématiques et de faire profiter au maximum chaque type de légume de l'éclairage et de la luminosité appropriés. Par exemple, placer un carré de maïs dont la hauteur dépasse un mètre face au

soleil privera les plantes des carrés situés derrière, de la lumière et de la chaleur dont elles ont besoin. La nature du sol est également une contrainte à prendre en compte. En effet, certaines plantes préfèrent une terre grasse et humide tandis que d'autres préfèrent un sol aride. La rotation des cultures qui consiste à ne pas semer deux plantes identiques ou de la même famille ou variété successivement sur un même carré afin d'éviter la prolifération des maladies ou des insectes n'est pas primordiale pour le jardinage en carrés. En effet, le fait que les carrés soient petits impliquent qu'une fois un légume récolté, sa saison est terminée, aussi le légume planté à sa suite sera forcément de type différent. Toutefois, la famille du légume est une propriété à considérer car certaines familles ne supportent pas d'être semées immédiatement après certaines autres. Enfin, et ceci constitue la dernière contrainte à prendre en compte concernant les légumes, il faut considérer les relations de compagnonnage. Nous avons déjà mentionné qu'il existait trois relations. La relation *amie* signifie que deux plantes forment une bonne association. La relation *ennemie* concerne les plantes dont la présence de l'une inhibe la croissance de l'autre. La relation *protectrice* est utilisée pour les plantes qui favorisent la croissance, améliorent le goût ou protègent des insectes d'autres plantes. Ces trois relations dirigent en fait le processus de résolution. Elles sont décisives lors du placement des plants de légumes dans le jardin. Nous venons d'énoncer les contraintes associées aux légumes pour les disposer dans un jardin. Celles-ci sont insérées en tant que caractéristiques des cas ***légume***. Nous pouvons constater ici qu'il s'agit du même cas ***légume*** que pour la classification auquel on a rajouté certaines propriétés utiles pour la planification.

Cas légume

- **n° d'identification :**
  numéro du cas
- **nom :**
  nom du légume;
- **famille :**
  chénopodiacées, composées, crassulacées, crucifères, cucurbitacées, labiacées, légumineuses, liliacées, ombelliféracées, papilionacées, solanacées, valérianacées, inconnue;
- **variété :**
  nom de plante dont elle partage la variété;
- **type de plante :**
  grimpante, rampante, arborescente, racine, ...;

- **compagnonnage :**
  - amies : liste de noms de plantes amies;
  - ennemies : liste de noms de plantes ennemies;
  - protectrices : liste de noms de plantes protectrices;
- **hauteur de la plante (en cm) :**

  0, 15, 30, 45, 90, 100, 200;
- **largeur (en cm) :**

7.5, 10, 15, 30, 60, 90, 120;
- **longueur (en cm) :**

7.5, 10, 15, 30, 60, 90, 120;
- **nombre de plants/carré :**

0.125, 0.25, 0.5,1, 2, 4, 8, 9, 16;
- **superficie** (surface occupée par un plant sur un carré en cm^2 = largeur*longueur) :

56.25, 100, 112.15, 225, 450, 900, 1800, 3600, 7200, 8100;

Exemple de cas légume pour la planification:

- *N° identification* 7
- *Nom* aubergine
- *Famille* cucurbitacées
- *Variété* légume
- *Type de plante* rampante
- *Hauteur* 60
- *Largeur* 30
- *Longueur* 30
- *Nombre de plants/carré* 1
- *Superficie* 900
- *Compagnonnage*
  - ➤ *amies* haricot à rame, haricot vert, piment
  - ➤ *ennemies*
  - ➤ *protectrices* fève verte

Pour configurer un jardin nous avons donc besoin tout d'abord des cas de type ***légume*** qui contiennent toutes les caractéristiques associées à chaque légume, mais il est également nécessaire de définir un cas ***jardin*** qui contient lui, la disposition des carrés dans le jardin. Comme nous l'avons décrit dans le modèle d'organisation, chaque cas ***jardin*** a été obtenu suite aux configurations déjà réalisées, à l'époque en format papier, et qui sont converties en cas en fonction des descripteurs de celui-ci. Pratiquement, pour représenter un cas jardin, il suffit d'énoncer la liste des légumes contenus dans un jardin et le résultat de la configuration

soit pour chaque carré le nom du légume qu'il contient. En effet, on peut considérer un jardin comme un tableau à deux dimensions tel que représenté ci-dessous:

	1	2	3	4
1	concombre	concombre	fèves	fèves
2	radis	ail	patate	patate
3	radis	trévise	trévise	menthe
4	thym	épinard	tomate	tomate

Cas jardin :

Problème :

- **n° identification :**

  identificateur de cas dans la base

- **liste_légumes :**

  liste de légumes pour un total de 16 plans avec pour chacun d'eux le nom et le nombre de plants. Chaque composant de cette liste est donc relié à la base de cas ***légume*** et aux descripteurs qui ont été mentionnés plus haut.

Solution :

- **configuration :**

  organisation des plants dans le jardin

Exemple de cas légume pour la planification:

- *N° identification* 2
- *Liste légumes* {(concombre 2) (fèves 2) (radis 2) (ail 1) (patate 2) (tomate 2) (épinard 1) (trévise 2) (thym 1) (menthe 1)}
- *Configuration* { (concombre concombre fèves fèves) (radis ail patate patate) (radis trévise trévise menthe) (thym épinard tomate tomate)}

Nous venons donc de définir toutes les connaissances qu'il est nécessaire d'acquérir pour procéder à la planification d'un jardin. Nous n'avons pas fait allusion aux contraintes et aux connaissances liées à la résolution de problèmes, à savoir sur la méthode même de configuration car elles ne sont pas traitées à ce niveau. Nous allons maintenant considérer le niveau des inférences.

b) Niveau des inférences

Comme pour la tâche de classification, la planification est une tâche du RPC. Aussi, elle est réalisée en suivant le même cycle en quatre phases du RPC qui sont retrouver, adapter, vérifier/valider et enregistrer d'où les quatre structures d'inférences.

Nous avons constaté dans le modèles des tâches que la tâche planification se décomposait en un ensemble de sous-tâches. Celles-ci concernent la recherche d'anciens plans déjà existants, la création de nouveaux plans, la validation et vérification de chaque plan trouvé ou créé, et finalement la plus importante car elle marque la différence avec la classification, l'adaptation des plans en fonction des besoins et des contraintes. Ceci explique les modifications apportées aux structures d'inférence de la classification.

- **Structure d'inférence de la tâche RECHERCHER (figure 15) :**

La structure d'inférence rechercher est composée de la SI de la tâche Recherchée définie dans le niveau inférence de la classification et de la SI de la tâche **valider** qui permet de vérifier et de valider que la configuration contenue dans le cas retrouvé est correcte, c'est-à-dire qu'elle respecte les contraintes que nous avons décrites au niveau du domaine. Les métaclasses impliquées sont *cas-sélectionné* en entrée et *cas-validé* et le résultat de la validation à savoir les *erreurs* trouvées, s'il y en a, en sortie. La figure 14 montre la nouvelle SI de la tâche valider.

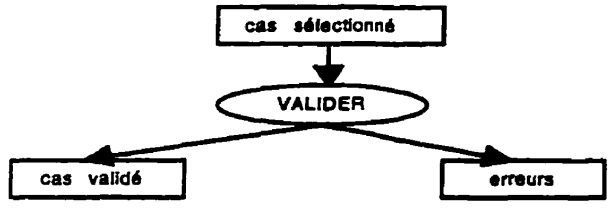

Figure 14. SI de la tâche *Valider*

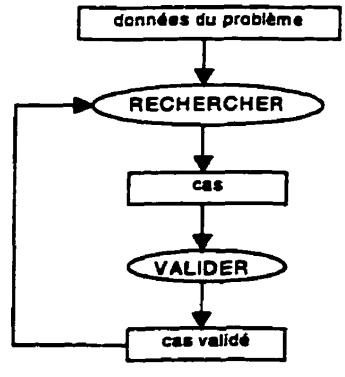

Figure 15. SI de la tâche *Rechercher* planification

SC **valider**

(Entrée	*cas-sélectionné*,
Sortie	*cas-validé* : cas transformé car il n'était pas valide,
	erreurs : la liste des incohérences trouvées dans le plan)
Méthodes	Algorithme de vérification dans l'arbre de décision;
Connaissances	Contraintes du domaine, règles de compagnonnage, de disposition.

- **Structure d'inférence de la tâche ADAPTER (figure 17):**

La structure d'inférence de la tâche **adapter** est composée de la SI de la tâche **adapter** de la classification et de la nouvelle SI **créer** qui est déclenchée quand aucun cas n'a été retrouvé, ou qu'aucun ne convient suffisamment pour être adapté. Ensuite le cas créé est aussi transformé en cas solution. La métaclasse produite est *cas-créé*. La figure 16 montre la SI de la tâche **créer**.

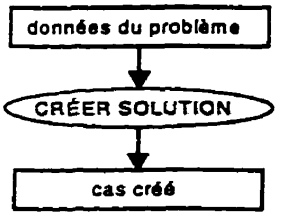

Figure 17. SI de la tâche *Créer*

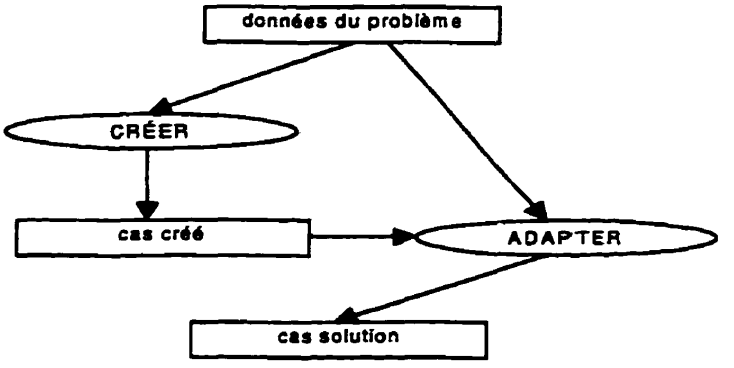

Figure 16. SI de la tâche *Adapter* planification

SC **créer-solution**

(Entrée	*données-du-problème,*
Sortie	*cas-créé* : selon les mêmes formalisme et principes que les cas déjà existants)
Méthodes	Algorithme de création, de configuration, de planification d'un jardin;
Connaissances	Du domaine, contraintes et règles de planifications, besoins des demandeurs.

- **Structure d'inférence de la tâche VÉRIFIER / VALIDER :**

C'est la même que pour la classification à la différence que pour la planification, la vérification et la validation sont effectuées par le système et non pas par l'utilisateur. Les algorithmes de vérification sont bien définis. La métaclasse *univers-des-observables* est donc remplacée par la métaclasse *modèle-du-système*.

- **Structure d'inférence de la tâche ENREGISTRER**

Elle est identique à celle que nous avons définie pour la classification.

c) Niveau des tâches

Le niveau des inférences n'ayant subi que peu de changements par rapport à celui de la classification, nous n'allons spécifier que les modifications propres à la planification. Ainsi, nous avons, ici aussi, quatre tâches qui correspondent aux tâches du RPC. Cependant, les

buts à atteindre prennent en compte les particularités associées à la tâche de synthèse de la planification.

tâche rechercher

but : rechercher un cas dont les valeurs des descripteurs en fassent un cas similaire au cas problème et qui soit correct avec le domaine

termes de contrôle :liste-erreurs : ensemble des erreurs trouvées

structure de tâche : rechercher (données du problème -> cas sélectionné)

```
{ abstraire (données-du-cas -> cas)
| indexer (base-de-cas, modèle-du-système -> structure indexée)
 comparer (cas, structure-indexée -> liste-de-cas)
 RÉPÉTER
     RÉPÉTER  sélectionner (liste-de-cas -> cas-sélectionné)
     TANT QUE pertinence-cas-retrouvé insuffisante
     valider(cas-sélectionné ->erreurs)
     RÉPÉTER
       valider(cas-sélectionné ->erreurs)
       mettre-à-jour (erreurs ->liste-erreurs)
     TANT QUE erreur = VRAI
     valider(cas-sélectionné -> cas-validé)
 TANT QUE refus=vrai}
```

tâche adapter

but : utiliser la solution du cas retrouvé ou créé pour résoudre le cas courant et adapter le cas ainsi trouvé afin qu'il réponde aux conditions de départ.

structure de tâche : adapter (données-du-problème, cas-similaire -> cas-solution)

```
{ instancier-solution(données-du-problème, cas-similaire -> cas-instancié) |
 créer-solution(données-du-problème -> enregistrement créé)
 RÉPÉTER
     transformer(cas | enregistrement-créé-> cas-solution)
 TANT QUE transformation-à-faire = vrai}
```

tâche vérifier

la seule différence vient du fait que la métaclasse de départ sur laquelle va être basée la vérification est le *modèle-du-système* et non pas l'*univers-des-observables*.

tâche enregistrer

Cette tâche n'a pas subi de modifications.

d) Niveau de stratégie

C'est à ce niveau que nous pouvons donner les contraintes et les contrôles sur la résolution de problème pour chaque tâche spécifiée. Les informations relatives à la communication sont aussi explicitées.

- **Rechercher** : l'utilisateur fournit la *liste des légumes* dont il veut composer le jardin. Le système va rechercher en *comparant* les cas qui possèdent le plus grand nombre de types de légumes en commun avec la liste initiale et fournit la liste des cas qui sont les plus proches de la configuration demandée, c'est-à-dire des cas qu'il a sélectionnés. Le système doit conserver les connaissances explicatives qui permettront à ce stade de générer des explications sur pourquoi un cas a été choisi plutôt qu'un autre, comment ont été retrouvé les cas, les critères de similarité, la méthode d'indexation utilisée, la pertinence des cas retrouvés, les ressemblances et différences avec le cas présenté afin d'aider le choix d'un cas précis qui sera ensuite vérifié par le système qui dressera la liste des erreurs qu'il a trouvées. La justification de celles-ci par le système influence les choix d'acceptation ou de refus du cas présenté.
- **Adapter** : si la configuration proposée convient, elle est réutilisée pour le cas courant. Si celle-ci ne convient pas à l'utilisateur, il peut soit modifier la composition des carrés lui-même et ainsi, à partir de l'exemple et de ce qu'il désire, créer un nouveau jardin, ou encore demander au système de le transformer en fonction de ses besoins. Des explications doivent être disponibles pour l'aider dans son choix afin qu'il évite au maximum des erreurs de configuration. Le système réalise une adaptation automatique dans le cas où aucune des configurations retrouvées n'a obtenu un score suffisant. Il construit alors lui-même une nouvelle configuration selon la stratégie de configuration qu'il a choisie. Les choix qu'il fait peuvent être décrits à l'aide des explications. Dans tous les cas d'adaptation, le système doit conserver la trace du raisonnement qu'il a suivi ainsi que toutes les étapes significatives pour les intégrer dans un descripteur spécifique du cas. Ces données seront réutilisées lors de l'explication de la trace du raisonnement.
- **Vérifier** : une fois que le système a adapté le cas, il doit valider les cas obtenus. S'il y a des erreurs, celles-ci doivent être identifiées et expliquées afin qu'elles puissent être corrigées soit par l'utilisateur, soit par le système en recommençant l'adaptation.

- **Enregistrer** : quand les étapes d'adaptation et de vérification sont terminées, l'utilisateur peut choisir d'intégrer le nouveau cas et sa solution dans la base de cas correspondante. Enfin, le système montre graphiquement la configuration finale à l'utilisateur.

Le niveau de stratégie nous a donc permis de reprendre les contraintes et les contrôles qui régissent la résolution de problème. Nous pouvons constater que la planification nécessite davantage de traitements que de classification. En effet, si la classification n'a pour rôle essentiel que de retrouver des classes spécifiques à l'aide de techniques inductives déjà existantes, la planification implique l'élaboration de techniques qui permettent d'adapter, de créer et de vérifier et ce, en fonction du domaine considéré. Le niveau de stratégie est spécifique car il marque l'introduction de la communication du module de planification avec le module d'explications. En effet, il spécifie à l'aide du niveau des tâches les types d'explications qui sont nécessaires en fonction des tâches et à quel moment elles doivent être générées. Ceci nous amène à considérer le modèle d'expertise des explications qui est traité dans le prochain chapitre. Nous avons donc dans ce chapitre procédé à la modélisation de la résolution de problème pour la classification et la planification, fonctions de notre application. Les modèles produits sont les modèles d'organisation, de tâches, d'agents et de communication ainsi que le modèle d'expertise pour la résolution de problème. Les connaissances explicatives contenues dans chacun de ces modèles vont servir de support pour élaborer le modèle d'expertise des explications.

CHAPITRE VI

Modélisation des explications et conception du système

Nous avons, dans le chapitre précédent, déterminé le modèle d'expertise spécifique au RPC que nous appellerons modèle d'expertise de résolution de problème (ME-RDP). Nous avons, pour chacune des tâches de résolution que nous traitons, décomposé le problème en tâches et en sous-tâches. Cependant, même si nous avons considéré les explications dans les différentes étapes de l'analyse de la classification et la planification, d'autres connaissances doivent être acquises pour élaborer un dispositif explicatif. En effet, nous avons identifié des besoins en explications, comme par exemple pour demander à expliquer les similarités trouvées ou les méthodes qui ont permis de retrouver des cas spécifiques. À chaque fois, ce sont des types d'explications qui sont considérés. Aussi, il est nécessaire d'étudier ce que nous appelons la typologie des explications et que nous avons évoquée dans le chapitre IV.2. Celle-ci fournit un support pour le modèle d'expertise des explications (ME-E) et permet de compléter les modèles déjà produits lorsque de nouvelles connaissances sont acquises. Nous allons donc dans ce chapitre définir le ME-E qui terminera la phase d'analyse selon KADS de notre application. Ensuite, dans la deuxième partie du chapitre, nous procéderons à la deuxième phase de la méthode qui est celle de la conception du système.

VI.1 Élaboration du Modèle d'expertise pour les explications

Afin de définir le ME-E, il est nécessaire de compléter l'étude de la typologie des explications, réalisée dans le chapitre IV, afin d'acquérir les connaissances explicatives nécessaires et de compléter les modèles produits dans le chapitre V. De plus, notre objectif étant de générer des explications dans les SBC-RPC en général, nous devons définir le contexte de la génération des explications dans un cadre spécifique aux SBC-RPC en nous basant sur

l'application de jardinage en carrés. Le résultat de cette étude préliminaire nous permettra d'élaborer le ME-E.

VI.1.1 Analyse générale de la typologie des explications

L'analyse de la typologie est réalisée dans un cadre général. Elle se situe au même niveau que le modèle générique d'explications que nous avons élaboré. Elle consiste à déterminer les types d'explications dont une application peut avoir besoin en fonction des différents critères du modèle générique du chapitre IV et les connaissances explicatives associées. C'est une analyse du domaine des explications.

L'étude de la typologie des explications représente un sujet vaste et complexe qui a fait l'objet de nombreux travaux et ce surtout pour les SBC [Bourcier *et al.* 94; Springer 93; Baumewerd-Ahlmann *et al.* 91b; Leake 95]. La particularité que nous nous proposons d'apporter ici est de réaliser cette typologie dans les SBC-RPC car elle est peu présente dans ce domaine. Le rôle d'un mécanisme de génération d'explications est de répondre aux interrogations des utilisateurs, qu'ils soient matérialisés par des personnes ou par une machine, et d'être capable de couvrir le plus grand espace de questions possibles. Ainsi, à chaque question identifiée correspond un type précis d'explication. On distingue essentiellement deux grandes classes d'explications (voir chapitre IV) : **l'explication des connaissances** et **l'explication du raisonnement** (ou justification). L'interprétation quant au type d'explication à générer est déterminée par le contexte. Cependant, en informatique il est difficile de faire interpréter un contexte à une machine, aussi, les mots clés interrogateurs permettent de comprendre les questions posées. Les quatre types d'interrogation auxquels nous nous intéressons sont : QUOI, POURQUOI, COMMENT et POURQUOI PAS. Chaque type réfère à un niveau d'abstraction, à des connaissances et à des méthodes différentes, et correspond à l'une ou l'autre des deux classes d'explications identifiées plus haut. Le traitement associé à la génération d'explications correspondant au type de question spécifiée, en accord avec la classe d'explication concernée, peut être considéré comme une tâche ou un but d'explications [Bourcier *et al.* 94]. Aussi, en considérant qu'une explication est un objet, elle doit appartenir à l'une des deux classes d'explications. Pour chacune d'elles, on doit identifier tout d'abord le type de question (ou d'explication demandée), puis le but d'explication à atteindre, suivi des connaissances statiques ou dynamiques qui sont impliquées, les stratégies explicatives

associées et finalement les techniques pour les élaborer. Il s'agit donc d'instancier les critères du modèle générique d'explication [Verrière 97]. Les éléments principaux de cette décomposition sont montrés dans la figure 18. Nous venons ainsi de dresser le modèle des tâches du dispositif explicatif avec sa décomposition en tâches et sous-tâches. Nous avons déjà, dans le chapitre V.2.1, décomposé la tâche d'explication en deux sous-tâches : explication du RPC et explication des connaissances générales. Cette décomposition est conforme à la typologie en deux classes que nous avons élaborée. Elle permet d'organiser les critères du modèle générique d'explications et les connaissances manipulérs. Nous allons dans ce qui suit définir les éléments qui ne l'ont pas encore été.

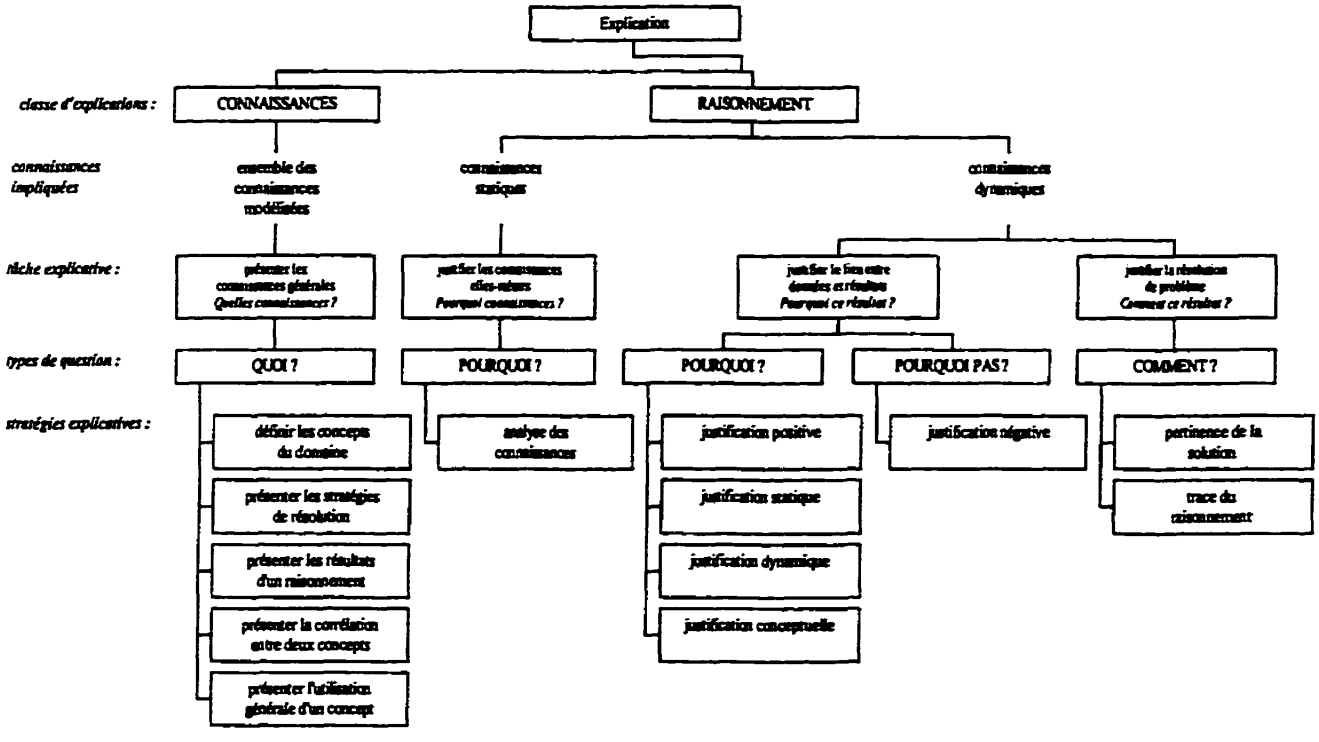

Figure 18. Décomposition de la tâche d'explication

VI.1.1.1 Typologie des questions

Nous allons définir les quatre types de questions que nous avons citées en évoquant pour chacune d'elles les connaissances et les stratégies explicatives auxquelles elles sont liées.

- *Quoi ?*

Ce type de question fait appel aux connaissances même du domaine d'application. Elles ne font pas appel au raisonnement utilisé mais aux concepts et aux relations liant ces concepts, et aux définitions. Elles nécessitent donc d'autres connaissances que celles utilisées pour la

résolution de problème. Elles agissent comme une sorte d'option d'aide disponible à tout moment par l'utilisateur. Elles doivent être retenues pendant l'acquisition des connaissances du domaine. Elles sont formulées sous forme d'écran d'aide, de graphisme, de patrons d'explications, de message prédéfinis ou d'images. Ces explications sont déterminées au niveau domaine du modèle d'expertise. Elles ont besoin des connaissances du cas courant et des connaissances en général de l'application.

exemple : *En quoi consiste l'application?*
Que signifie que deux plantes sont amies ou ennemies ?
Qu'est-ce qu'une plante de la famille des Liliacées?

- *POURQUOI ?*

Ce type de question entre dans la catégorie des justifications des connaissances. Elles ne font pas appel au type de résolution de problème utilisée par le système. Ce sont des explications mettant en évidence des relations de cause à effet. Elles interviennent quand le résultat a été donné, au niveau des inférences du modèle d'expertise de résolution de problème. Elles nécessitent l'accès aux métaclasses d'entrées impliquées afin de récupérer les valeurs de celles-ci comme réponses.

exemple 1: Pourquoi deux légumes sont-ils juxtaposés ?

Il faut utiliser les règles de compagnonnage entre les plantes. Les relations impliquées ici sont plantes amies, plantes protectrices, plantes indifférentes (cette relation est déduite pour deux légumes quand il n'ont aucune relation de compagnonnage en commun).

exemple 2: pourquoi utiliser 6 plans pour les tomates ?

Il faut faire appel à la relation de superficie.

exemple 3: pourquoi cette configuration ?

Il faut dresser les relations de compagnonnage pour l'ensemble du jardin, nombre de plants/carré. Ici il faut aussi faire appel non plus aux simples connaissances, mais il est également nécessaire d'utiliser des méthodes propres à la résolution telle que la méthode de vérification de jardin décrite dans le niveau de tâche du modèle d'expertise de résolution de problème.

- *POURQUOI PAS ?*

Ce type de question est très populaire et est très proche des questions qui peuvent être posées dans la vie courante. Il faut utiliser le niveau des inférences du modèle d'expertise de la résolution de problème en fournissant les métaclasses qui sont impliquées. S'il faut justifier la

non exécution d'une tâche alors c'est le niveau tâche qui doit être utilisé avec les buts et les contraintes et les règles de contrôle impliquées dans la tâche.

exemple : Pourquoi ne pas avoir mis le brocoli à côté des tomates?

Il faut là aussi utiliser les relations de compagnonnage, mais cette fois, il faut justifier par le concept ennemi, ou alors, si rien ne justifie signaler qu'il n'y a pas de contre-indication à semer ces deux types de légume dans deux carrés voisins. Une autre réponse consiste à visualiser ce qui se passe dans les autres carrés et justifier par rapport à ceux-ci. Un légume peut ne pas avoir été semé dans un carré voisin d'un autre légume même s'ils sont amis parce qu'il est ennemi d'un autre type de légume avec lequel il se retrouverait en contact aussi, plusieurs carrés se touchant dans un jardin.

- *COMMENT ?*

Ce type de question fait appel à la méthode de résolution utilisée, soit aux connaissances dynamiques [Scott&Weckert 97]. Dans le cas du RPC, la méthode à expliquer est celle du processus qui a permis de retrouver les cas, il faut donc donner les règles de similarités utilisées. Dans notre cas, la règle de similarité utilisée pour la configuration est le nombre maximum de légumes apparaissant dans le cas retrouvé et faisant partie de la liste initiale. Pour la classification, la similarité est déterminée par le parcours des branches d'un arbre de décision en calculant le résultat obtenu sur chacune d'elle.

exemple : Comment le système a-t-il retrouvé cette configuration?

réponse : En retrouvant 5 légumes en commun avec la liste demandée.

Une autre explication à une question de type *COMMENT* concerne la trace du raisonnement. Ainsi, le système doit redonner à l'utilisateur l'ensemble des étapes suivies par le système depuis la donnée du problème jusqu'à l'élaboration de l'ensemble des solutions. En fait il faut instancier les étapes du RPC avec les données du problème.

Une dernière explication à une question de type *COMMENT* peut intervenir concernant le processus utilisé par le système pour créer une solution quand il n'en trouve pas de suffisamment intéressante.

exemple : Comment le système a-t-il trouvé cette configuration ?

Le système doit alors expliquer les différentes étapes qu'il a suivies pour trouver une solution qui convienne aux souhaits de l'utilisateur. Ici, c'est la ligne d'explication [Paris *et al.* 88] qui

doit être fournie à l'utilisateur à moins que l'utilisateur veuille connaître les tentatives échouées du système pour trouver un voisin à un légume spécifique. Donc, on peut fournir deux explications ici [Paris *et al.* 88]:

- la ligne du raisonnement suivie par le système : elle indique pas à pas la résolution du système, et ce de façon détaillée.
- la ligne d'explication : celle-ci fait abstraction des problèmes spécifiques à la résolution de problème et est plus proche des besoins de l'utilisateur. Elle fait appel à un ensemble de connaissances qui ne sont pas utilisées par le processus de résolution.

Ainsi, l'ensemble des questions que l'utilisateur peut poser couvre les différents types d'explication identifiés lors de l'analyse.

VI.1.1.2 Autres composants

Les autres constituants de la décomposition ont déjà été décrits dans le modèle générique du chapitre IV.

Nous venons donc dans cette partie de définir la typologie des explications telle que nous la concevons. Celle-ci permet d'organiser les critères du modèle afin de procéder à l'élaboration des explications. Nous devons maintenant établir le contexte dans lequel cette typologie évolue par rapport au modèle d'expertise de résolution de problème.

VI.1.2 Modèle d'expertise d'explications/résolution de problèmes

Notre approche repose sur le fait que la génération d'explications doit être traitée comme une tâche de résolution à part entière avec ses propres connaissances et ses propres techniques de résolution de problèmes. C'est pourquoi nous avons choisi de construire un Modèle d'Expertise spécifique aux Explications (ME-E). Celui-ci a la même structure que le Modèle d'Expertise spécifique à la Résolution De Problème (ME-RDP) et chaque niveau du ME-E englobe la connaissance du ME-RDP faisant du ME-RDP une sous-partie du ME-E. Le ME-RDP contient à chaque niveau la connaissance de résolution de problème correspondante. Aussi, certaines de ces connaissances sont utiles à la génération d'explication. C'est pourquoi, l'approche généralement suivie consiste soit à identifier à chaque niveau du ME-RDP le type de question qui peut être traitée, soit à déterminer les niveaux du modèle qui vont être utiles à la résolution d'une question [Sprenger 93; Baumewerd-Ahlmann et al. 91a]. Ils identifient

pour chaque type de question les métaclasses ou les sources de connaissances impliquées. Aussi, leur approche ne consiste-t-elle donc pas vraiment à modéliser selon la méthode KADS la tâche de génération d'explications mais plutôt à extraire du ME-RDP les connaissances explicatives utiles à la génération d'explications. Ces connaissances explicatives ont été identifiées lors de l'étude du ME-RDP que nous avons faite. Celles-ci ne sont toutefois pas suffisantes. Il manque les stratégies de construction d'explications et les connaissances utiles à la génération d'explications qui ne sont pas représentées dans le ME-RDP. De plus, les connaissances relatives à la résolution de problèmes pour un cas précis ne sont pas conservées explicitement. C'est pourquoi il apparaît nécessaire de créer un modèle spécifique au cas traité (MSCT) tel qu'il a été introduit par Bourcier [Bourcier *et al.* 94]. Nous avons déjà évoqué l'utilité de celui-ci dans la section IV.2.8. Celui-ci doit intégrer toutes les connaissances générales du cas qu'elles soient propres au domaine ou propre à la résolution de problème du cas même et des résultats obtenus. C'est Bourcier qui a introduit l'utilité d'un tel modèle. Il décompose le ME-RDP en trois niveaux. Le premier niveau contient la connaissance de contrôle (appelée théorie de contrôle TC) qui correspond à la connaissance dynamique acquise des 3 niveaux du ME-RDP (inférences, tâches et stratégie), ensuite il décompose le niveau du domaine du ME-RDP qui contient l'ensemble des connaissances statiques du domaine en deux modèles, le modèle du domaine (ou théorie du domaine TD) qui contient les concepts et les relations du domaine, donc les connaissances générales du domaine, et un nouveau modèle qui est un modèle spécifique du cas traité (MSCT). Ce dernier contient les données initiales du problème donné, les hypothèses et les solutions trouvées. Cette approche est très intéressante du point de vue du RPC, car le fait de représenter toute la connaissance spécifique à un problème donné dans un même cas permet de représenter directement ce MSCT. Nous avons donc déjà défini ce modèle dans le niveau du domaine du ME-RDP avec les cas de type ***légume*** et ***jardin***. Le modèle d'expertise spécifique aux explications doit lui aussi être décomposé selon ces trois niveaux. Ainsi, le premier niveau qui correspond au MSCT contient la connaissance du MSCT du ME-RDP ainsi que les traces du raisonnement et toutes les connaissances de résolution qui ont permis d'aboutir au résultat du cas traité. Le RPC offre l'avantage de pouvoir intégrer toute cette connaissance dans un même cas, nous en donnerons la représentation dans la section VI.4. Le deuxième niveau, qui contient la théorie du domaine, intègre les 2 niveaux du domaine et de contrôle du ME-RDP auxquels on rajoute la modélisation des connaissances du domaine

utiles pour les explications. Finalement, le dernier niveau qui correspond à la théorie du contrôle, est propre aux explications et ne contient que les méthodes et connaissances relatives aux stratégies explicatives. La figure 19 montre l'organisation du ME-E telle que nous venons de la définir et les relations, les échanges de connaissances entre les modèles d'expertise de la résolution de problème et des explications, et la correspondance avec le modèle d'expertise tel qu'il est défini dans KADS. Ceci permet d'identifier la nature des connaissances explicatives associées aux différents niveaux du ME-E.

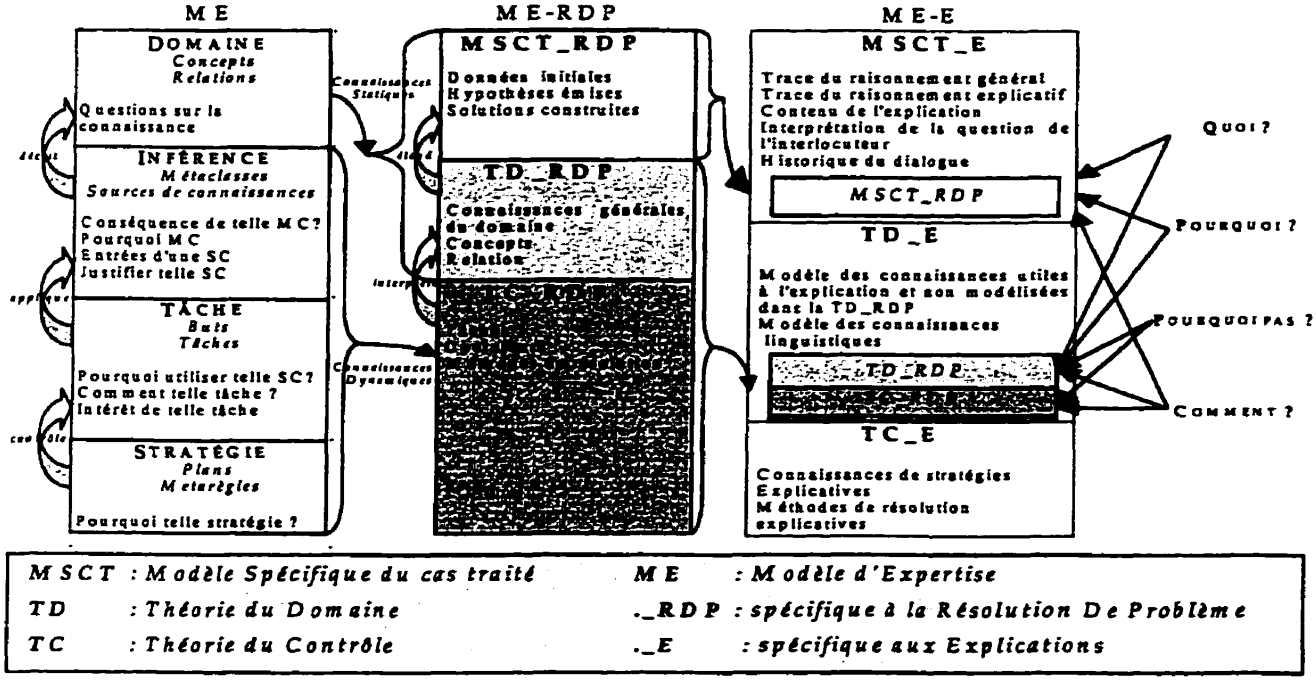

Figure 19. Organisation des modèles d'expertise inspiré de [Bourcier *et al.* 94; Springer 93]

Nous avons montré quelle devait être la composition du ME-E par rapport au ME-RDP que nous avons défini en V.4. Ceci nous a permis de montrer la réelle utilité de modéliser explicitement les connaissances et techniques explicatives et à quel niveau de ME. Afin de faciliter cette modélisation, il est nécessaire de compléter le modèle de communication afin d'établir les relations de communication et de coopération entre le module de RDP du RPC et celui des explications.

VI.1.3 Modèle de communication pour les explications

La description du modèle de communication n'a pas été entièrement définie dans le paragraphe V.4. En effet, elle nécessite une meilleure connaissance des besoins en explications conformément à la méthode de RDP choisie. Les différents éléments de contrôle

de la communication entre le système et l'utilisateur ont été énumérés dans le modèle d'expertise de la RDP à chaque étape du raisonnement au fur et à mesure des besoins. Le RPC étant un raisonnement suivant un cycle de processus spécifique, chaque phase fait appel à des connaissances et fournit des résultats différents qui peuvent nécessiter une demande d'explication. Aussi les différents besoins en explications qui ont été décrits dans les paragraphes précédents sont repris de manière générale dans la figure 20. Ils permettent ainsi de faire une généralisation des besoins en explications dans le processus de résolution des SBC-RPC et donc de montrer la communication entre les différents agents intervenant dans l'application soit le système avec la RDP, l'utilisateur et le module explicatif. La demande d'explications est réalisée soit par le système soit par l'utilisateur comme nous l'avons décrit dans le modèle de l'agent (paragraphe V.3).

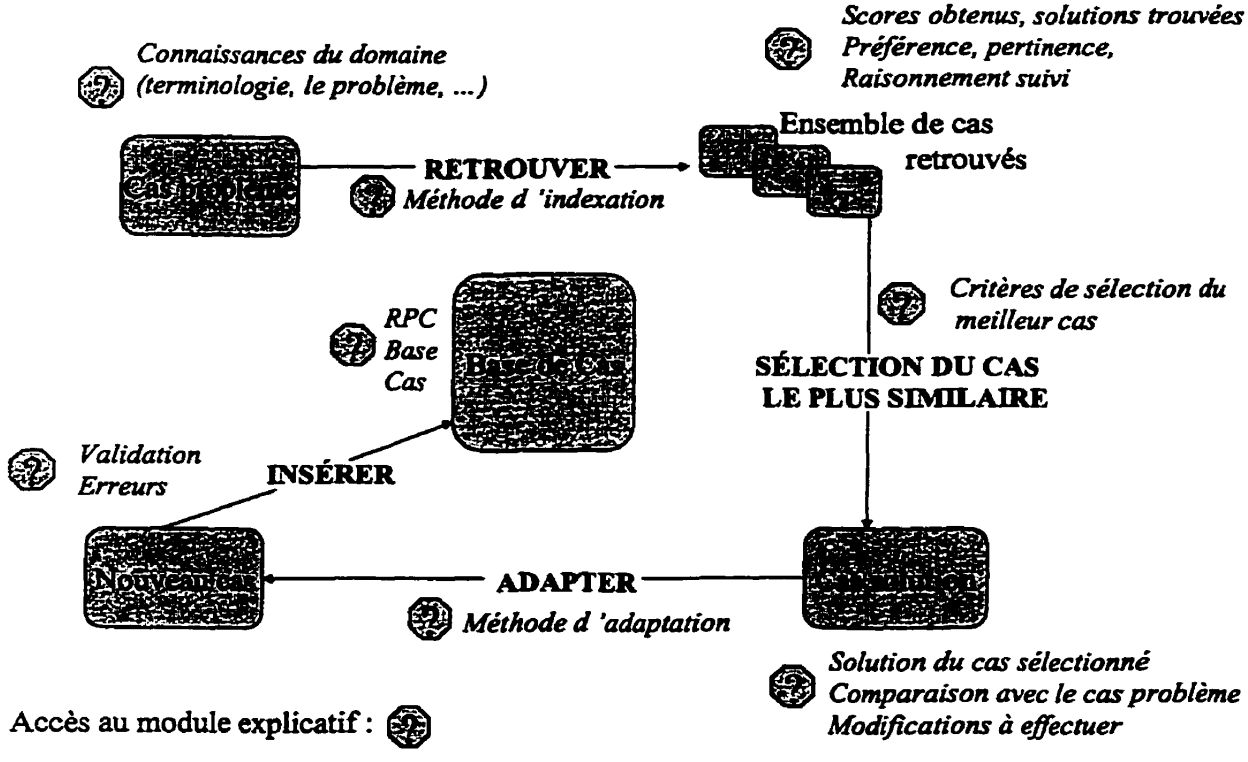

Figure 20. Modèle de communication pour les explications dans les SBC-RPC

Nous avons dans les parties précédentes défini la typologie des explications, l'organisation et la nature des connaissances qui doivent composer le ME-E et finalement la communication entre les différents agents et modules du système pour satisfaire les besoins en explications.

Nous possédons donc maintenant tous les éléments nécessaires pour procéder à l'élaboration du ME-E.

VI.1.4 Modèle d'expertise

Le modèle spécifique aux explications a donc la même structure que celle du modèle de la résolution de problème. Néanmoins, à la différence de Bourcier [Bourcier *et al.* 94], qui élabore un niveau spécifique dans le ME pour traiter le MSCT, nous intégrons ce modèle au niveau du domaine car il est représenté par l'intermédiaire des cas qui sont représentés dans ce niveau. Il n'est donc pas nécessaire d'élaborer un niveau supplémentaire pour spécifier le cas traité. Nous décrivons dans cette section les quatre niveaux du modèle d'expertise.

VI.1.4.1 Niveau du domaine

Le niveau du domaine du ME-E a pour rôle de définir les concepts et les relations du domaine intervenant dans l'élaboration des explications. Il contient la connaissance statique définie dans le ME-RDP car elle représente l'information de base sur le domaine pour élaborer le contenu des explications. Mais ce niveau doit aussi contenir d'autres connaissances qui ne sont pas utilisées lors de la RDP telles que les définitions du domaine, les principes de la méthode, etc.

La première catégorie de connaissances à modéliser dans le niveau du domaine concerne donc les connaissances générales du domaine qui ne sont pas utilisées pendant la résolution de problème. Ce sont les connaissances qui sont implicites pour l'expert du domaine. Elles sont, pour cette raison, difficiles à acquérir car elles apportent de l'information à l'utilisateur non familier avec les connaissances manipulées mais sont acquises pour un expert. On identifie dans cette catégorie de connaissances les définitions du domaine, celles ayant trait à la RDP et les définitions générales des connaissances du domaine.

- **définitions du domaine**

Exemple :

"Le jardinage en carrés est une nouvelle façon de disposer, de planter et d'entretenir un jardin productif et attrayant. ...".

"La classification permet de classifier les légumes selon leur famille d'appartenance, en fonction d'un ensemble de descripteurs sur leur forme végétative, leur couleur, etc, ... à l'aide du RPC. ..."

- **définitions sur la RDP**

Exemple :

"Le Raisonnement Par Cas ou RPC est un mode de raisonnement basé sur l'analogie qui suit un cycle de résolution de problème en quatre phases : retrouver, réutiliser, réviser, retenir, ..."

"Le processus retrouver permet de rechercher dans la base de cas les cas les plus similaires au cas problème, ..."

Ces deux types de définitions sont prédéfinis, ils correspondent à du texte déjà rédigé. Ils ne respectent pas de structure précise. Ils décrivent textuellement les méthodes de résolution. Ils permettent la création d'un module d'aide tel qu'on peut en trouver dans tous les systèmes traditionnels.

- **définitions générales des connaissances du domaine**

Exemple :

"La famille des chénopodiacées est une famille de plantes sauvages aux usages médicinaux, répandues près des vieux murs. Exemples: bette, betterave, épinard"

"Deux plantes sont amies quand elles forment une bonne association. Exemple : la betterave est l'amie du chou"

Chacune des définitions de ce type est caractérisée par :

- Un nom : terme à définir
- Une définition : définition associée au terme
- Des exemples : exemples permettant d'illustrer la définition du terme

La deuxième catégorie de connaissances explicatives concerne les connaissances statiques utilisées lors de la résolution de problème. Ces sont les connaissances du niveau du domaine du ME-RDP. Elles correspondent aux cas de types ***légume*** et ***jardin*** tels qu'ils ont été définis.

À cela on doit ajouter des descripteurs supplémentaires dans les cas ***légume*** et ***jardin*** qui ne sont pas utiles pour la RDP mais servent en tant que connaissances explicatives. Ces descripteurs supplémentaires sont :

- **Image** : permet de donner une représentation graphique d'un légume ou d'un jardin
- **Forme** : donne un supplément d'information quant à la description générale d'un légume
- **Origine** : pays d'où vient le légume

Les connaissances explicatives suivantes à considérer sont celles qui appartiennent au MSCT. Elles consistent d'une part à la représentation instanciée du cas traité selon le formalisme des cas ***légume*** et ***jardin***. D'autre part, elles concernent les connaissances issues de la résolution de problème du cas traité. Il s'agit de :

- **Solution** :
  famille d'appartenance du légume pour la tâche de classification
  plan configuré de jardin en carrés pour la tâche de planification
- **Trace du raisonnement** : les différentes étapes de résolution de problème qu'a suivies le système pour aboutir à la solution. C'est l'ensemble des règles déclenchées avec les faits impliqués et les valeurs ou nouveaux faits trouvés
- **Résultats intermédiaires** : ensemble de paires attributs-valeurs qui correspondent à des résultats intermédiaires qui permettent soit de compléter les descripteurs manquants ou d'apporter un supplément d'information utilisable dans les explications

Ces connaissances, si elles ne le sont pas déjà, sont représentées à l'intérieur même des cas traités.

La dernière catégorie concerne les connaissances explicatives sur la résolution de problème générale telle qu'elle est appliquée par le système. Elle est contenue dans ce qu'on appelle :

- **Arbre de résolution général** : contient l'ensemble des connaissances de contrôle du ME-RDP, à savoir les informations sur la RDP contenues dans les niveaux inférence, tâche et stratégie

Le niveau du domaine du ME-E est donc composé de deux types de connaissances : celles qui sont directement issues du ME-RDP, que ce soit pour les connaissances statiques ou pour la RDP même, et celles qui n'interviennent pas dans le cadre de la RDP mais qui agissent comme complément d'informations pour créer des explications adaptées aux besoins des utilisateurs.

VI.1.4.2 Niveau des inférences

Nous allons dans ce niveau montrer les inférences qui composent la tâche générale d'explication telle que nous l'avons définie dans le modèle générique d'explications et conformément à la typologie des explications décrite en VI.1.

a) Modèle de tâches associé aux explications

Nous avons défini dans le chapitre V.2 le modèle de tâches de notre application. La tâche d'explication y est décomposée en deux sous-tâches : expliquer le RPC et expliquer les connaissances. Ici, nous réalisons la décomposition globale de la tâche d'explication en prenant en compte toutes les sous-tâches qu'elle implique. Le modèle résultant montre les tâches associées au traitement des critères du modèle générique d'explications du chapitre IV. Ceci nous permet d'une part de compléter le modèle de tâches et d'autre part cette étape est nécessaire pour pouvoir identifier les inférences qui composent la tâche d'explication. L'arbre de décomposition de la tâche d'explication en sous-tâches est donné dans la figure 21.

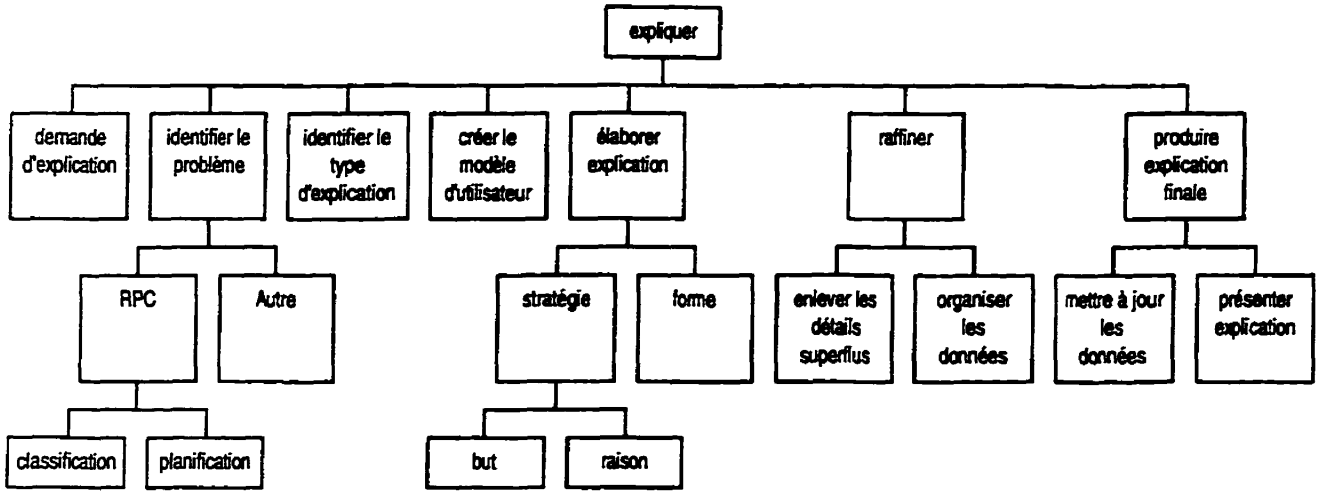

Figure 21. Décomposition de la tâche d'explication

b) Structures d'inférence

Cette décomposition de la tâche **expliquer** permet de déterminer les structures d'inférences du ME-E et de définir les SC et les MC impliquées.

- Structure d'inférence de la tâche **IDENTIFICATION** (figure 22): elle permet, à partir des données de l'utilisateur et des patrons de question déjà existants, d'identifier les données de la question qui vont servir pour élaborer l'explication.

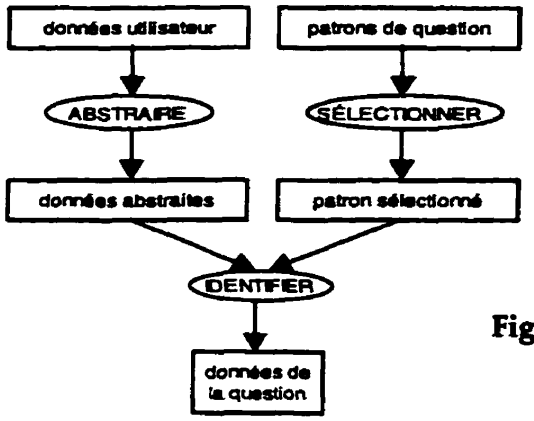

Figure 22. SI de la tâche *Identification*

SC **abstraire**

(Entrée	*données-utilisateur,*
Sortie	*données-abstraites* : en fonction des descripteurs d'une question type)
Méthodes	Méthodes d'abstraction
Connaissances	

SC **sélectionner**

(Entrée	*patrons-de-question* : ensemble de patrons types de questions
Sortie	*patron-sélectionné* : patron type choisi par l'utilisateur)
Méthodes	Méthode de sélection, choix utilisateur
Connaissances	Critères de sélection

SC **identifier**

(Entrée	*données-abstraites,*
	patron-sélectionné
Sortie	*données-de-la-question:* données pertinentes en fonction du patron de la question et des données disponibles
Méthodes	Identification des données utiles, instanciation de patrons
Connaissances	Signification des descripteurs de patrons

- Structure d'inférence de la tâche **ANALYSE** (figure 23) : elle permet d'analyser les données de la question afin de définir son modèle qui contient le type de problème traité (connaissance ou RDP), le type d'explication demandée en fonction des quatre types de questions (quoi, pourquoi, pourquoi pas et comment) et le modèle de l'utilisateur.

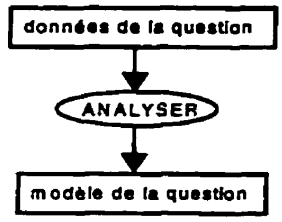

Figure 23. SI de la tâche *Analyse de la question*

SC **analyser**

(Entrée	*données-de-la-question*
Sortie	*modèle-de-la-question* : qui contient le type de problème, le type d'explication et le modèle de l'utilisateur)
Méthodes	Méthodes d'analyse, d'identification de type de problème, de type d'explication, et d'élaboration du modèle de l'utilisateur
Connaissances	Connaissances de l'utilisateur, sur les types de problème et types d'explication existants

- Structure d'inférence de la tâche **GÉNÉRER** (figure 24): elle permet d'élaborer le contenu de l'explication à partir du modèle de la question et d'une stratégie d'explication.

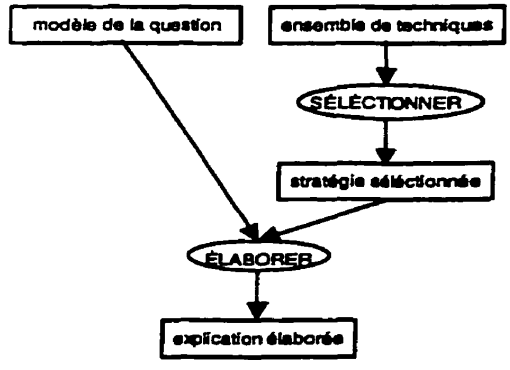

Figure 24. SI de la tâche *Générer des explications*

SC **sélectionner**

(Entrée	*ensembles de techniques* : stratégies de génération d'explications
Sortie	*stratégie-sélectionnée* : qui contient la méthode à suivre pour générer une explication)
Méthodes	Méthodes de sélection de techniques en accord avec les contraintes et les données disponibles
Connaissances	Permettant de sélectionner une technique appropriée

SC **élaborer**

(Entrée	*modèle-de-la-question*
	stratégie-sélectionnée
Sortie	*explication-élaborée* : sous le formalisme de représentation approprié et conformément au but désiré

Méthodes	Élaboration d'explications, représentations des explications
Connaissances	Connaissances explicatives et du ME-RDP et stratégiques explicatives nécessaires

- Structure d'inférence de la tâche **RAFFINER** (figure 25): elle permet d'organiser les données et de raffiner l'explication élaborée, en enlevant les détails superflus, en fonction du modèle de l'utilisateur et d'un ensemble de normes prédéfinies sur les explications.

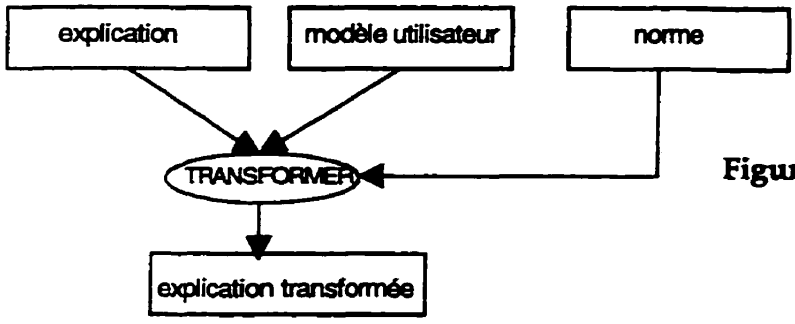

Figure 25. SI de la tâche *Raffiner*

SC **transformer**

(Entrée	*explication*
	modèle-de-l'utilisateur : contenant ses besoins, ses préférences, son niveau de compétence
	norme: de présentation et d'organisation des données d'une explication
Sortie	*explication-transformée* : conformément aux besoins)
Méthodes	Transformation d'une structure existante en fonction d'un ensemble de facteurs, méthodes d'adaptation
Connaissances	Contraintes, besoins, normes

- Structure d'inférence de la tâche **ENREGISTRER** (figure 26) : elle permet de présenter à l'utilisateur l'explication finale et d'insérer celle-ci ou les modifications qu'elle a entraînées dans le cas qui en est à l'origine.

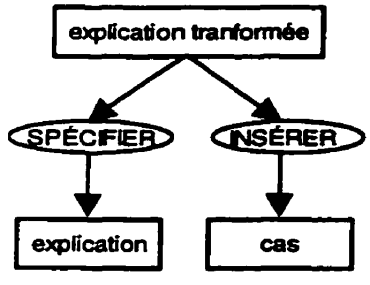

Figure 26. SI de la tâche *Enregistrer*

SC **spécifier**

(Entrée	*explication transformée*
Sortie	*explication* :finale telle qu'elle apparaît à l'utilisateur)
Méthodes	Méthodes de présentation, gestion des interfaces
Connaissances	Interfaçage, communication homme-machine

SC **insérer**

(Entrée	*explication transformée*
Sortie	*cas*
Méthodes	De modification du contenu d'un cas, contraintes de changement
Connaissances	Contenu du cas

La tâche d'explication peut donc être décrite par les cinq SI que nous venons de définir. Nous allons maintenant étudier le niveau des tâches.

VI.1.4.3 Niveau des tâches

À partir des SI que nous avons identifiées pour chaque sous-tâche de la tâche d'explication, nous pouvons déterminer les buts qu'elles doivent atteindre en fonction des contraintes et des structures de contrôles. Le niveau des tâches est donc composé des cinq tâches : identifier, analyser, générer, raffiner et enregistrer qui sont décrites ci-dessous.

tâche identification

but : identifier une structure de question qui corresponde à un patron de question déjà existant et conformément aux données qui doivent être expliquées.

structure de tâche : identification (données-utilisateur -> données-de-la-question)

```
{ abstraire (données-utilisateur -> données-abstraites)
  RÉPÉTER
      sélectionner(patrons-de-question ->patron-sélectionné)
  TANT QUE trouvé = FAUX
  identifier(données-abstraites, patron-sélectionné -> données-de-la-question)}
```

tâche analyser

but : analyser les données de la question afin de dresser un modèle de celle-ci en identifiant e type de problème, le type de question et le modèle de l'utilisateur. Identifier aussi les buts et raisons de l'explication (modèle générique d'explication).

structure de tâche : analyse (données-de-la-question -> type-problème, type-question, modèle-utilisateur)

{ analyser (données-de-la-question -> modèle-de-la-question)}

tâche générer

but : générer l'explication demandée conformément aux données disponibles sur l'utilisateur, et sur la question demandée et sous la forme pertinente sachant qu'à une technique correspond une représentation spécifique.

structure de tâche : générer (modèle-de-la-question -> explication-élaborée)

{ RÉPÉTER

sélectionner(ensemble-de-techniques -> stratégie-sélectionnée)

TANT QUE trouvé = FAUX

élaborer(modèle-de-la-question, stratégie-sélectionnée -> explication-élaborée)}

tâche raffiner

but : fournir une explication qui n'est pas chargée de détails superflus en accord avec le modèle de l'utilisateur et dont le contenu est organisé.

Structure de contrôle : superflus : ensemble de détails superflus

structure de tâche : raffiner (explication -> explication-transformée)

{ RÉPÉTER

transformer(explication, modèle-utilisateur, norme -> explication-transformée)

TANT QUE superflus = non vide}

tâche enregistrer

but : fournir à l'utilisateur l'explication finale, conserver une trace de l'explication dans le cas si c'est possible et si c'est utile pour une future utilisation

structure de contrôle : trace : signifie qu'il faut conserver la trace dans le cas

structure de tâche : enregistrer (explication-transformée -> explication-finale)

{ spécifier (explication-transformée -> explication)

si trace alors insérer(explication-transformée -> cas)}

VI.1.4.4 Niveau de stratégie

La stratégie utilisée pour la génération d'explications est de respecter la prise en compte des critères définis dans le modèle générique d'explications, d'acquérir et d'ordonner les informations les caractérisant pour obtenir des explications qui soient les plus

compréhensibles et les plus adaptées aux besoins des utilisateurs tout en se préoccupant des problèmes ergonomiques afin de les rendre conviviales. Chaque tâche à exécuter pour générer des explications permet d'accomplir ce que nous avons appelé dans le chapitre IV une stratégie explicative. C'est pourquoi, le niveau de stratégie pour le ME-E ne contient pas beaucoup d'éléments de stratégie. Chaque explication peut être considérée par sa spécificité comme une stratégie pour répondre à une question.

Dans cette partie, nous avons donc défini le modèle d'expertise spécifique aux explications. Celui-ci a la particularité de modéliser tous les types d'explications qui peuvent avoir à être générés à un moment quelconque d'une résolution de problème. Il est composé de ses propres connaissances et stratégies explicatives avec une méthode de résolution de problème spécifique à la tâche d'explication. De plus, il fait appel aux connaissances contenues dans le ME-RDP car elles apportent les connaissances nécessaires pour expliquer des résultats trouvés, le raisonnement utilisé ou même les connaissances manipulées. Aussi, à chaque niveau du ME-RDP peut être associé un type particulier d'explication. Nous avons précisé les types d'explications qui peuvent être élaborés et la tâche globale d'explication qui leur est associée. La phase d'analyse étant terminée, nous allons maintenant, à partir du modèle conceptuel composé des ME-RDP, ME-E et du modèle de communication, procéder à la phase de conception.

VI.2 La phase de Conception du système

Le chapitre V et la première partie du chapitre VI ont consisté en la modélisation de la phase d'analyse selon la méthode KADS. Le résultat de cette phase repose sur l'élaboration de six modèles : les modèles de l'organisation, des tâches, de l'agent, de communication et de deux modèles d'expertise, le ME-RDP et le ME-E. L'ensemble formé par le modèle de communication et les deux modèles d'expertise représente le modèle de résolution de problème adapté au problème traité et sert donc de base pour commencer la phase suivante de la méthode: la phase de conception. Celle-ci permet de spécifier d'une part l'architecture fonctionnelle du système et d'autre part l'architecture physique de celui-ci. Elle ne prend toujours pas en compte les problèmes d'implémentation du système. Néanmoins, c'est au cours de cette phase que les méthodes et paradigmes d'IA qui seront utilisés sont choisis de même que les outils d'implémentation.

Comme nous l'avons mentionné dans le chapitre III, la phase de conception consiste à modéliser le système à partir des éléments définis lors de la phase d'analyse. Elle est composée de trois étapes. La première consiste à définir l'architecture fonctionnelle du système, la deuxième a choisir les méthodes à appliquer à chaque fonction du système pour la résolution de problème. Celle-ci est représentée dans le modèle de comportement. L'issue de cette étape permet de définir l'architecture physique du système. Finalement, la dernière étape doit permettre de faire le choix des logiciels et de l'environnement utilisés pour l'implémentation du système.

VI.2.1 Le modèle fonctionnel

Le modèle fonctionnel a pour rôle, à partir des modèles fournis à l'issue de la phase d'analyse, de définir l'architecture fonctionnelle du système et de décrire les fonctions que devra comporter le système.

La figure 27 donne l'architecture du système telle qu'elle a été conçue à partir du modèle conceptuel.

L'architecture fonctionnelle de notre système est donc composée de quatre blocs fonctionnels. Le premier, **interface usager,** a pour rôle de gérer la communication entre le système et l'utilisateur par le biais de menus, d'écrans de saisie, d'écrans de résultats. Le deuxième contient les fonctions liées à la résolution de problème. On distingue deux sous-blocs pour la RDP, un pour la tâche de classification et un pour la tâche de planification. Le troisième bloc s'occupe de la gestion des cas de la base de cas générale. Celle-ci est constituée des cas utilisés pour la planification, pour la classification et pour les explications. Finalement le dernier bloc est le module d'explication. Il permet de générer l'ensemble des explications nécessaires à l'application. Nous allons décrire dans ce qui suit une partie de l'ensemble des blocs fonctionnels selon un ensemble de propriétés qui montrent les contrôles, tâches et données impliqués dans chaque bloc. Ainsi nous détaillerons les blocs qui sont d'une plus grande importance.

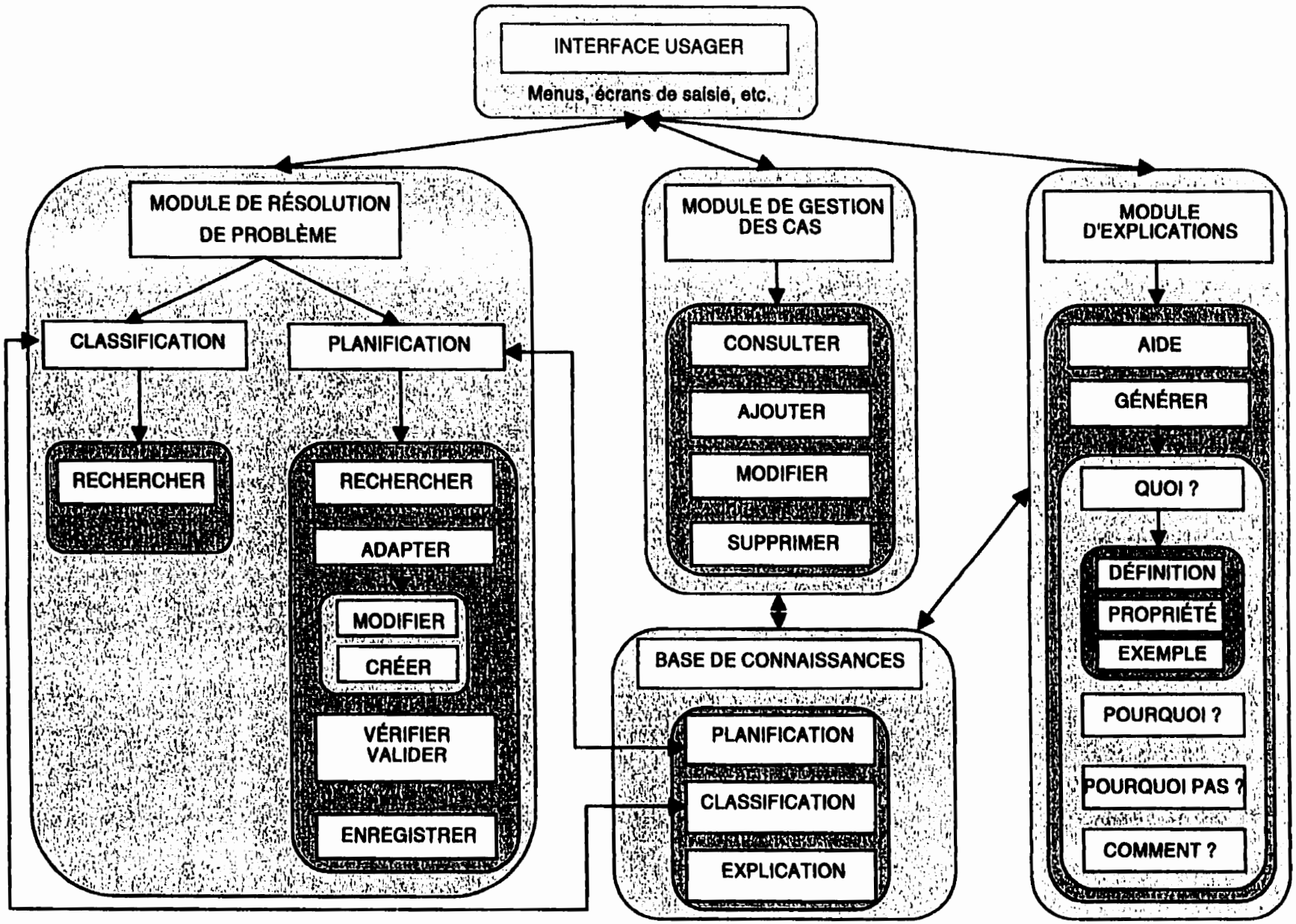

Figure 27. Architecture Fonctionnelle du Système

- *Bloc* **Interface Usager**

Est sous-fonction de :
Type de fonction : Interface utilisateur
Description : Permet d'assurer la communication entre le système et l'usager (demande de résolution de problème, demande d'explication, etc.)
Données en entrée : Valeurs des descripteurs d'un cas ou éléments d'une question
Données en sortie : Résultats de la RDP ou explication
Est contrôlé par : Début d'exécution, résolution de problème, acquisition des connaissances, génération d'explications
Contrôle : Validation par l'utilisateur
Lien avec le modèle conceptuel : Données
Est composé de : Menus, écrans de saisie, d'affichage

- *Bloc* **MODULE D'EXPLICATIONS**

Est sous-fonction de :
Type de fonction : Explication
Description : Permet de générer des explications en fonction du type de l'utilisateur et de l'explication demandée
Données en entrée : Éléments d'une question
Données en sortie : Explication
Est contrôlé par : Début d'exécution, résolution de problème, demande d'explication
Contrôle : Validation de la demande d'explication
Lien avec le modèle conceptuel : Structures d'inférences du niveau des inférences du ME-E
Est composé de : **AIDE, GÉNÉRER**

- *Bloc* **AIDE**

Est sous-fonction de : **MODULE D'EXPLICATION**
Type de fonction : Explication
Description : Affiche des écrans d'aide pour expliquer les fonctionnalités, le mode d'emploi, la terminologie comme dans un système traditionnel
Données en entrée : Écran, touche clavier (F1)
Données en sortie : Écrans d'aide
Est contrôlé par : touche clavier, option disponible sur l'écran en cours
Contrôle : Validation de sortie
Lien avec le modèle conceptuel : Tâche *expliquer les connaissances*
Est composé de :

- *Bloc* **GÉNÉRER**

Est sous-fonction de : **MODULE D'EXPLICATION**
Type de fonction : Interface utilisateur
Description : Présente sous forme de menu la liste des explications du raisonnement que l'on peut choisir
Données en entrée : Liste des choix possibles
Données en sortie : Un des choix sélectionné
Interface externe : Générateur de menu à option
Contrôle : Choix dans le menu, validation d'exécution
Lien avec le modèle conceptuel : Tâche *expliquer le raisonnement* , SI *Identifier*
Est composé de : **QUOI, POURQUOI, POURQUOI PAS, COMMENT**

- *Bloc* **QUOI**

Est sous-fonction de : **GÉNÉRER**
Type de fonction : Interface utilisateur

Description : Présente sous forme de menus les types d'explications des connaissances que l'on peut demander
Données en entrée : Liste des choix possibles
Données en sortie : Un des choix possibles
Interface externe : Menu à choix multiples
Contrôle : Sélection d'un type
Lien avec le modèle conceptuel : Tâche *expliquer les connaissances* (SI *analyser*), sélection d'une stratégie d'explication (SI *générer*)
Est composé de : DÉFINITION, PROPRIÉTÉ, EXEMPLE

- *Bloc* **POURQUOI**

Est sous-fonction de : GÉNÉRER
Type de fonction : Explication du raisonnement
Description : Permet de générer des explications sur pourquoi un résultat a été trouvé
Données en entrée : Un résultat
Données en sortie : Une explication sous forme d'un graphe de décision
Est contrôlé par : Données du problème, résultat trouvé, RDP
Contrôle : Résultat expliqué
Lien avec le modèle conceptuel : SC *élaborer* de la tâche *expliquer le raisonnement* avec modèle de la question définie
Est composé de :

- *Bloc* **POURQUOI PAS**

Est sous-fonction de : GÉNÉRER
Type de fonction : Explication du raisonnement
Description : Permet de générer des explications sur pourquoi un résultat n'a pas été trouvé
Données en entrée : Un résultat attendu
Données en sortie : Une explication montrant sous forme d'un graphe la différence entre le résultat attendu et celui obtenu
Est contrôlé par : Résultat obtenu, RDP, résultat attendu
Contrôle : Pertinence du cas retrouvé par rapport au cas attendu expliqué, ligne explicative
Lien avec le modèle conceptuel : SC élaborer de la tâche *expliquer le raisonnement* avec modèle de la question définie
Est composé de :

- *Bloc* **COMMENT**

Est sous-fonction de : GÉNÉRER
Type de fonction : Explication du raisonnement
Description : Permet de générer des explications sur comment un résultat a été trouvé
Données en entrée : Liste des cas retrouvés
Données en sortie : Une explication montrant sous forme d'un graphe les étapes du raisonnement suivies par le système lors de la RDP en fonction des données du domaine.
Est contrôlé par : Lancement de la RDP
Contrôle : Graphe expliquant la trace du raisonnement
Lien avec le modèle conceptuel : SC *élaborer* de la tâche *expliquer le raisonnement* avec modèle de la question défini
Est composé de :

- *Bloc* **MODULE DE GESTION DES CAS**

Est sous-fonction de :

Type de fonction : Interface utilisateur
Description : Gérer la consultation, l'ajout, la modification et la suppression des cas dans la Base de connaissances
Données en entrée : Description d'un cas
Données en sortie : Un cas
Est contrôlé par :
Contrôle :
Lien avec le modèle conceptuel : Primitive *indexer* de la SI de la tâche *Rechercher* du ME-RDP
Est composé de : CONSULTER, AJOUTER, MODIFIER, SUPPRIMER

- *Bloc* MODULE DE RÉSOLUTION DE PROBLÈME

Est sous-fonction de :
Type de fonction : Interface utilisateur
Description : Choisir entre deux types de RDP, planification ou classification
Données en entrée : Choix du menu
Données en sortie : Choix sélectionné
Interface externe : Menu principal
Contrôle : Type de RDP choisie
Lien avec le modèle conceptuel : Choix de la tâche de RDP à réaliser, tâche de *planification* ou de *classification*
Est composé de : CLASSIFICATION, PLANIFICATION

- *Bloc* CLASSIFICATION

Est sous-fonction de : MODULE DE RÉSOLUTION DE PROBLÈME
Type de fonction : RDP
Description : Permet de rassembler les données utiles pour la RDP
Données en entrée : Description du problème
Données en sortie :
Est contrôlé par :
Contrôle :
Lien avec le modèle conceptuel : tâche de *classification*
Est composé de : RECHERCHER

- *Bloc* RECHERCHER

Est sous-fonction de : CLASSIFICATION
Type de fonction : RDP
Description : Recherche les cas similaires au problème
Données en entrée : Descripteurs instanciés du cas
Données en sortie : Liste de solution
Est contrôlé par : Données du problème
Contrôle : Solution retrouvée
Lien avec le modèle conceptuel : Primitive *indexer* et *comparer* de la SI de la tâche *rechercher* du ME-RDP
Est composé de :

- *Bloc* PLANIFICATION

Est sous-fonction de : MODULE DE RÉSOLUTION DE PROBLÈME
Type de fonction : RDP
Description : Permet de rassembler les données utiles pour la RDP
Données en entrée : Données du problème
Données en sortie :
Est contrôlé par :
Contrôle :
Lien avec le modèle conceptuel : Tâche de *planification*

Est composé de : **RECHERCHER, ADAPTER, VÉRIFIER/VALIDER, ENREGISTRER**

- *Bloc* **RECHERCHER**

Est sous-fonction de : **PLANIFICATION**
Type de fonction : RDP
Description : Recherche des cas similaires au problème
Données en entrée : Descripteurs instanciés du cas
Données en sortie : Liste de cas
Est contrôlé par : Données du problème
Contrôle : Validation de l'utilisateur
Lien avec le modèle conceptuel : SI de la de tâche *rechercher* du ME-RDP
Est composé de :

- *Bloc* **ADAPTER**

Est sous-fonction de : **PLANIFICATION**
Type de fonction : RDP
Description : Permet en fonction des cas retrouvés ou des exigences de l'utilisateur de transformer le cas pour correspondre aux données et contraintes
Données en entrée : Un cas retrouvé
Données en sortie :
Est contrôlé par : Cas retrouvé non satisfaisant
Contrôle :
Lien avec le modèle conceptuel : SI de la tâche *adapter* du ME-RDP
Est composé de : **MODIFIER, CRÉER**

- *Bloc* **MODIFIER**

Est sous-fonction de : **ADAPTER**
Type de fonction : RDP
Description : Permet d'adapter la solution du cas retrouvé en fonction des données du problème
Données en entrée : Cas retrouvé avec sa solution
Données en sortie : Cas résolu
Est contrôlé par : Rejet de la solution
Contrôle : Solution modifiée
Lien avec le modèle conceptuel : Primitive *transformer* de la SI de la tâche *adapter* du ME-RDP
Est composé de :

- *Bloc* **CRÉER**

Est sous-fonction de : **ADAPTER**
Type de fonction : RDP
Description : Permet de créer une nouvelle solution
Données en entrée : Données du cas
Données en sortie : Solution
Est contrôlé par : Données du cas, rejet des solutions trouvées, aucune solution trouvée
Contrôle : Solution créée
Lien avec le modèle conceptuel : SC *créer* de la SI de la tâche adapter du ME-RDP
Est composé de :

- *Bloc* **VÉRIFIER/VALIDER**

Est sous-fonction de : **PLANIFICATION**
Type de fonction : RDP
Description : Évalue la solution proposée afin de vérifier qu'elle respecte bien les contraintes et les données du problème

Données en entrée : La solution
Données en sortie : La solution validée
Est contrôlé par : La solution du problème
Contrôle : La solution validée
Lien avec le modèle conceptuel : Les SI *vérifier* et *valider* avec la SC *transformer* du ME-RDP
Est composé de :

- *Bloc* ENREGISTRER

Est sous-fonction de : PLANIFICATION
Type de fonction : Interface utilisateur, gestion des cas
Description : Permet de présenter la solution finale et de mettre à jour la Base de connaissances
Données en entrée : Le cas résolu
Données en sortie :
Est contrôlé par : Le cas résolu
Contrôle : Validation de l'utilisateur
Lien avec le modèle conceptuel : SI *enregistrer*
Est composé de :

VI.2.2 Le modèle de comportement

Ce modèle décrit les méthodes qui réalisent les fonctions du système. Il découle de la description des blocs fonctionnels contenus dans l'architecture fonctionnelle. La hiérarchie des blocs de celle-ci permet d'identifier quatre méthodes principales : interfaçage, explication, gestion des données et, finalement, de résolution de problème avec les deux méthodes de planification et de classification.

- **Méthode d'interfaçage**

Description : interagir avec l'utilisateur pour faciliter la communication utilisateur-machine. Les méthodes utilisées doivent favoriser la convivialité en tenant compte des aspects ergonomiques qu'un système doit contenir pour permettre une manipulation facile.
Associée_à :INTERFACE USAGER, MODULE DE GESTION DES CAS
Éléments de conception : menus, langage naturel, question/réponse, écrans de saisie

- **Méthode d'explication**

Description : méthode permettant à l'utilisateur de demander des explications au système à tout moment pendant le processus de résolution de problème ou dès le départ pour se familiariser avec l'environnement.
Associée_à : Module d'Explication, Quoi, Définition, Propriété, Exemple, Pourquoi, Pourquoi Pas, Comment.

Éléments de conception : cas traité, mémoire de faits, bases de cas, procédures de manipulation des données existantes, de génération d'explications. Celles-ci comprennent : les procédures de stratégies explicatives : quoi, pourquoi, pourquoi pas, comment ainsi que les procédures de création d'explications : patron d'explications générique ou instancié, schéma, graphe, texte-prédéfini, tableau récapitulatif, ...

L'organisation de l'ensemble des éléments de conception qui interviennent dans les méthodes utilisées pour générer des explications est montrée dans les représentations graphiques de la conception des types d'explications quoi, pourquoi, pourquoi pas et comment (figures 28 à 31).

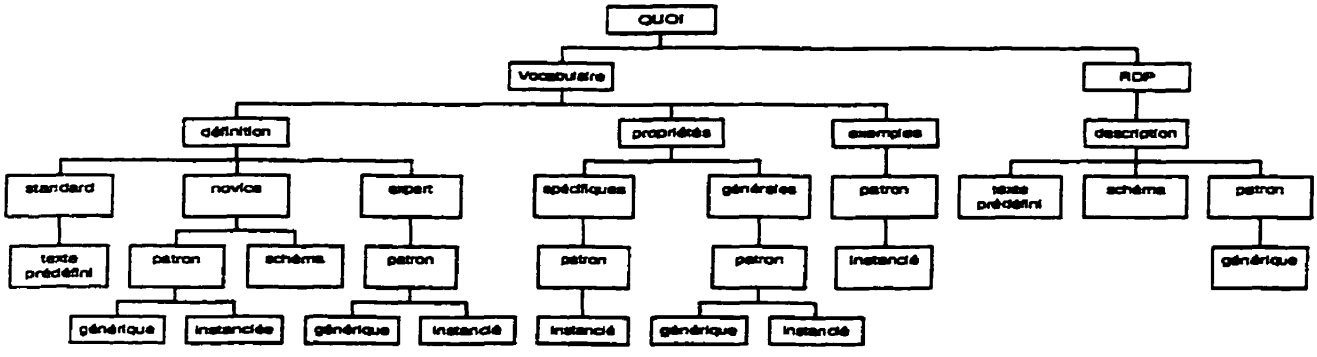

Figure 28. Éléments de conception de la question *Quoi ?*

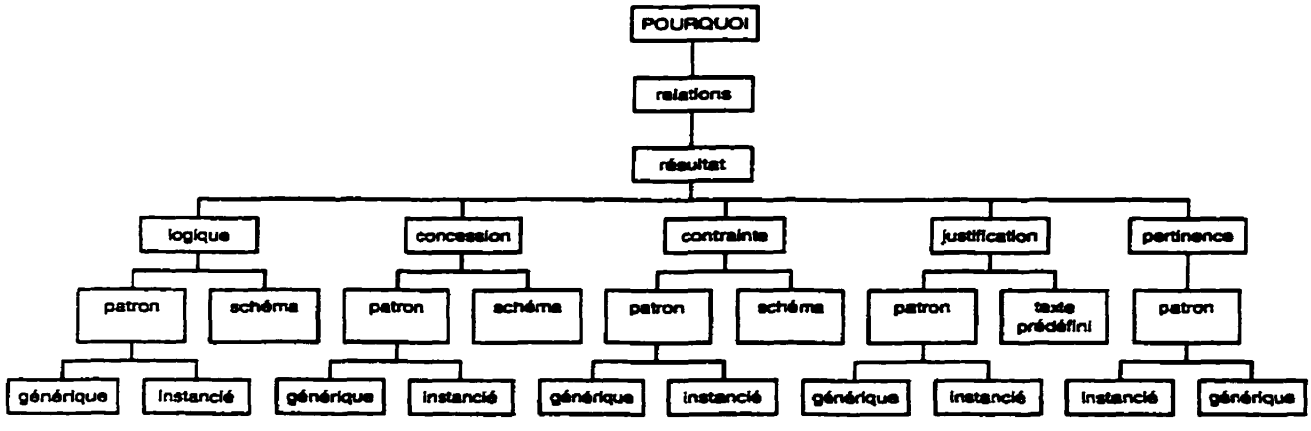

Figure 29. Éléments de conception pour la question *Pourquoi ?*

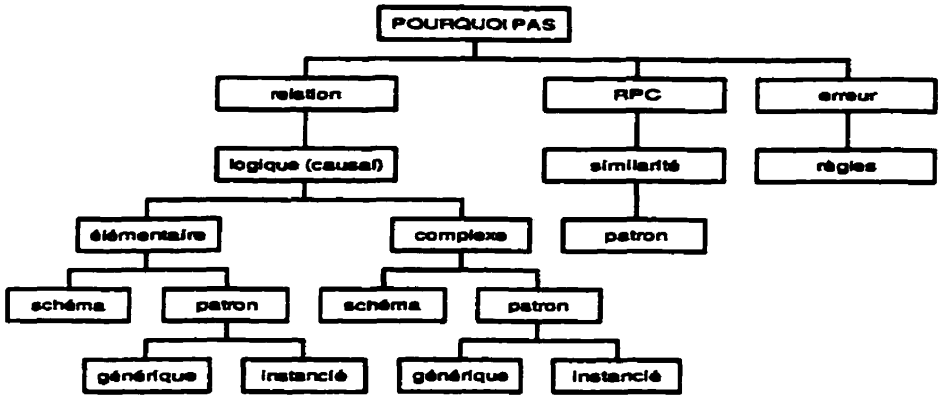

Figure 30. Éléments de conception de la question *Pourquoi Pas ?*

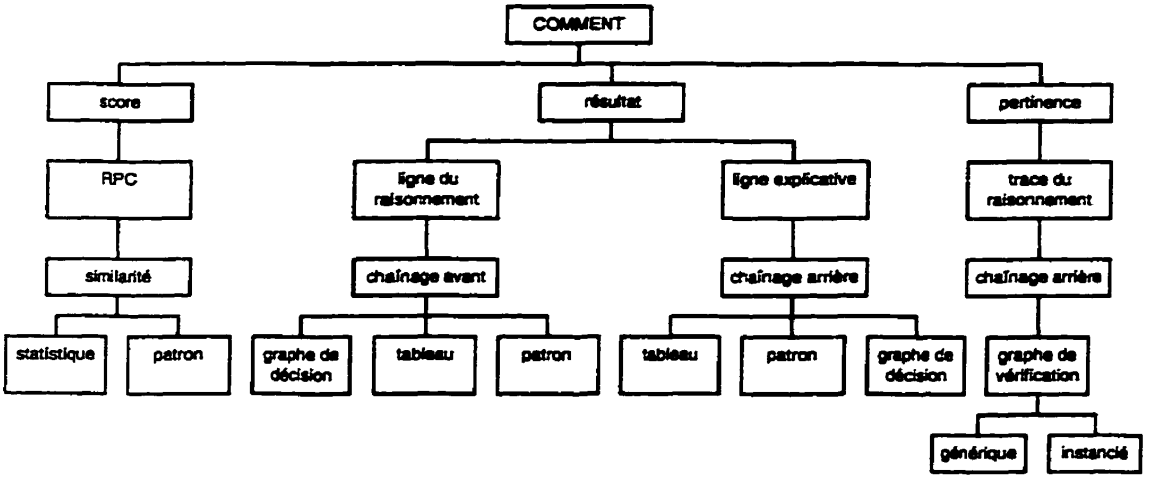

Figure 31. Éléments de conception de la question *Comment ?*

- **Méthodes de gestion des données**

Description : celles-ci vont permettre à l'utilisateur d'accéder aux bases de cas lors de la création de la base pour la phase d'acquisition des connaissances, de la résolution de problème ou pour la phase de maintenance de la base pour consulter, ajouter, modifier ou supprimer des cas.

Associée_à : Consulter, Ajouter, Modifier, Supprimer

Éléments de conception : mémoire de cas, base de données, des procédures de consultation, d'ajout, de modification et de suppression d'enregistrements dans une base. Il faut disposer de dispositifs de recherche de cas suffisamment efficaces pour retrouver rapidement des cas dans un base existante, d'où la nécessité de procédures d'indexation.

- **Méthode de classification**

Description : méthode de RDP utilisant la méthode du RPC pour réaliser la tâche de classification.

Associée_à : rechercher

Éléments de conception : procédures de manipulation de cas, d'indexation et de recherche efficace dans une structure indexée différentes des méthodes de gestion des cas.

- **Méthode de planification**

Description : méthode de RDP utilisant la méthode du RPC pour réaliser la tâche de planification.

Associée_à : Rechercher, Adapter, Vérifier/Valider, Créer, Enregistrer

Éléments de conception : procédures de manipulation de cas, d'indexation et de recherche efficace dans une structure indexée différentes des méthodes de gestion des cas et de celle utilisée pour la classification. Il doit aussi y avoir des méthodes de vérification de contraintes, de création de solution et de modification des solutions trouvées.

VI.2.3 Le modèle physique

Ce modèle est constitué d'un ensemble de modules qui spécifient comment les méthodes définies dans le modèle fonctionnel sont organisées de manière logique en fonction des éléments de l'architecture fonctionnelle pour constituer l'architecture physique du système qui sera implantée.

VI.2.3.1 Architecture physique

La figure 32 montre l'architecture physique du système qui doit être implanté. L'organisation des modules de RDP et de gestion de cas n'est pas détaillée. Nous ne montrons ici que l'architecture physique du module d'explication et les interactions avec les autres modules.

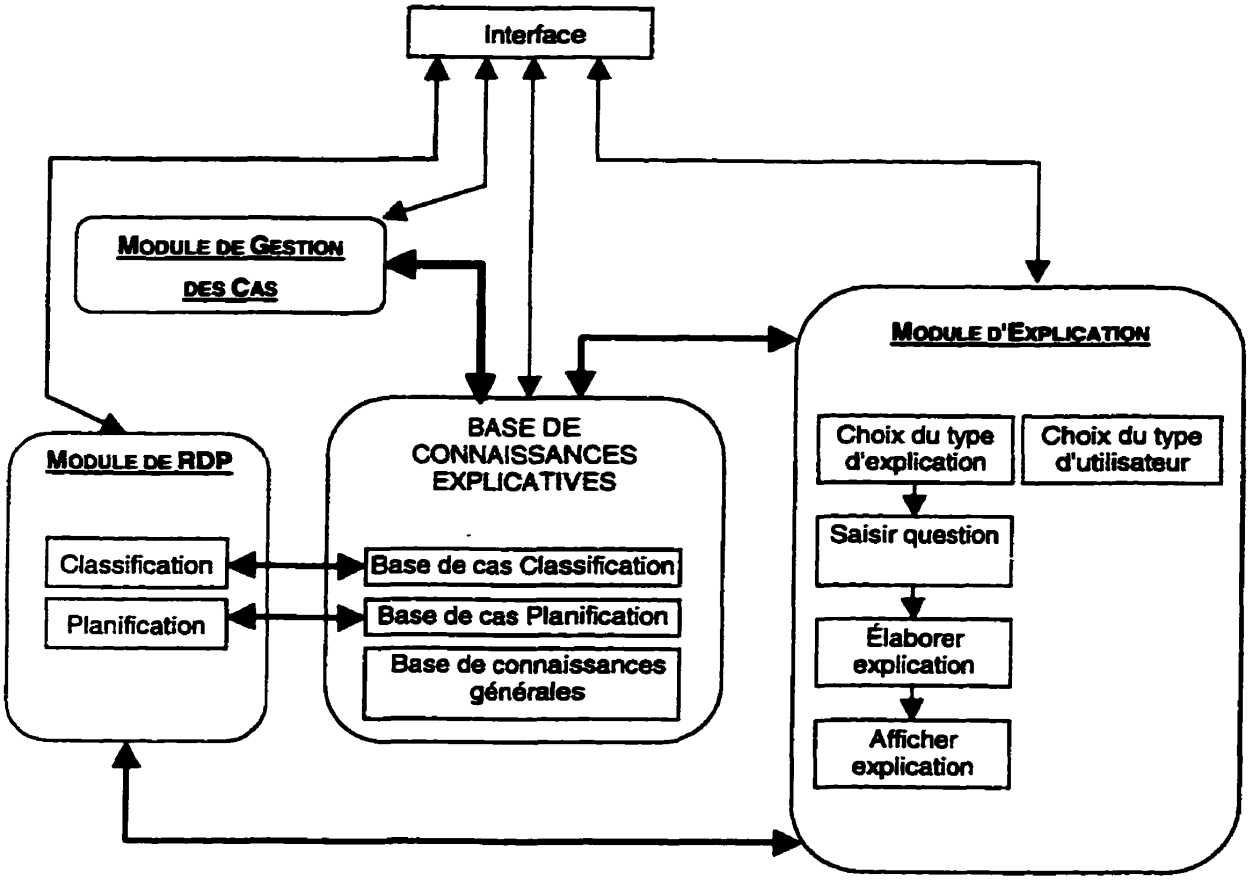

Figure 32. Architecture physique du système

VI.2.3.2 Description de l'environnement et des outils choisis

Nous décrivons ici les outils utilisés pour implémenter l'architecture proposée. Le système a été développé sur plate-forme PC et sous l'environnement Microsoft® Windows 95. Chaque module de RDP ou non nécessitant des traitements de nature différente, plusieurs outils ont été choisis pour implémenter chacun d'eux.

Nous avons choisi d'utiliser Delphi™ 3 de la société Borland® pour monter et gérer les interfaces ainsi que pour implémenter le moteur général de l'application qui permet de gérer les échanges entre les différents modules. Cet outil est un environnement de programmation visuelle en Pascal Orienté Objet utilisant la technologie RAD (Rapid Application Development). Il offre plusieurs avantages dont :

- un environnement de développement intégré qui supporte le développement, les tests et la maintenance de l'application;

- la connectivité par le biais de DLL à d'autres environnements de programmation comme Eclipse et Easy Reasoner;
- la connectivité à des bases de données intégrées permettant un accès transparent aux bases de données locales Paradox et dBase et au serveur InterBase ainsi que des traitements SQL;
- un compilateur de code natif qui crée un exécutable autonome et optimisé;
- un langage de programmation orienté objet;
- un environnement de programmation ergonomique.

Pour implémenter le module de raisonnement par cas pour la classification nous avons choisi d'utiliser Easy Reasoner™ de The Haley Enterprise, un outil de raisonnement par cas intégré à Eclipse, un logiciel à base de règles permettant de développer des systèmes experts. Easy Reasoner permet :

- d'utiliser les bases de données DBF pour structurer et stocker les cas;
- de faire une analyse statistique automatique des modèles de données;
- de retrouver des cas similaires de la mémoire à partir d'un nouveau cas;
- d'indexer des bases de données existantes en utilisant des arbres de décision;
- de supporter les bases de données xBase, ODBC et SQL;
- ranger et retrouver des cas classifiés par similarité dans de grandes bases de données;
- de classifier toute nouvelle information en utilisant les graphes de décision;
- classifier automatiquement et interactivement de nouveaux cas;
- supporte les données manquantes.

Eclipse™ est un système de production de grande performance qui fournit une version étendue de la syntaxe des langages à base de règles et qui inclut l'Agent OCX pour l'intégration COM(Common Object Model), le développement RAD et l'utilisation des DLL pour les bases de données et la connection à d'autres environnements de programmation comme Delphi. Éclipse permet de représenter facilement la connaissance et le raisonnement sous forme de règles, d'où son utilité pour générer des explications. De plus il intègre Easy Reasoner rendant facilement accessible toute la connaissance de la résolution de problème pour générer des explications.

Les bases de données utilisées sont au format DbaseIV et ont été créées avec ACCESS de Microsoft.

L'outil Microsoft Help Workshop a été choisi pour implémenter le module d'aide car il permet de créer facilement des fichiers d'aide. De plus, Delphi offre la possibilité d'intégrer les fichiers d'aides créé avec cet outil.

Ceci termine la phase de conception selon KADS de l'application de jardinage en carrés et du module d'explications que nous avons défini. Nous avons donc dans ce chapitre fini la phase d'analyse en présentant le ME-E propre à la génération d'explications dans les SBC-RPC. Puis, nous avons conçu les architectures fonctionnelles et physiques de notre système et montré l'environnement qui a été choisi pour le développer. Les deux phases principales de la méthode KADS ainsi réalisées, nous pouvons procéder à la phase d'implémentation du prototype qui fait l'objet du chapitre VII.

CHAPITRE VII

Expérimentation : prototypage et discussion

L'objectif de ce chapitre consiste tout d'abord à décrire le prototype EDEN que nous avons implanté à partir de l'architecture physique du système décrite dans le chapitre précédent. Ensuite, nous montrerons sur un exemple le fonctionnement d'EDEN. Nous terminerons ce chapitre par une discussion sur les résultats obtenus.

VII.1 Prototypage

L'objectif de la phase de prototypage est de permettre de valider les modèles élaborés dans les phases antérieures. Le prototype consiste en la mise en œuvre des fonctionnalités du système à l'aide des outils choisis lors de la phase de conception. Dans cette section nous allons tout d'abord présenter le prototype EDEN, ensuite nous décrirons les modules de résolution de problème pour la classification, de gestion des cas et finalement d'explications.

VII.1.1 Description d'EDEN (générateur conçu pour Élaborer Des Explications à partir d'un modèle géNérique)

EDEN est un prototype de SBC-RPC qui intègre des capacités explicatives, programmé dans un environnement composé d'un ensemble d'outils de développement. Il résulte de l'implémentation de l'application de jardinage en carrés que nous avons décrite dans les chapitres V et VI. Toutefois, à cause des limites évidentes d'un projet de maîtrise, celle-ci n'a pas été réalisée dans sa totalité. Le module de RDP pour la configuration n'a pas été implanté. Nous allons décrire le système avec les composants et les fonctionnalités qu'il contient actuellement (figure 33), à savoir : le module de gestion des cas (Base de cas), le module de classification de légumes et le module d'explication. Comme nous l'avions mentionné à

plusieurs reprises dans ce mémoire, l'objectif principal était de développer un système qui soit le plus convivial possible, ceci représentant déjà un gros avantage pour les questions de facilité d'utilisation et de compréhensibilité.

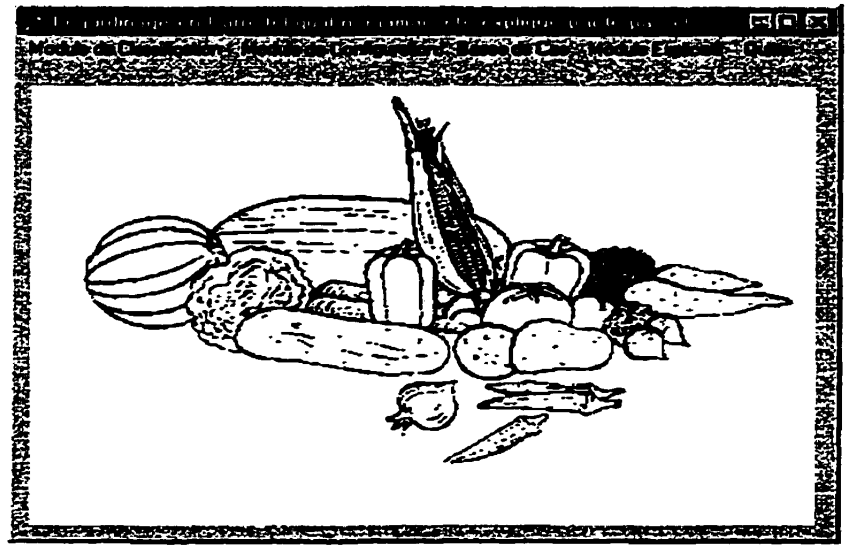

Figure 33. Interface principale d'EDEN

Le prototypage d'EDEN a été réalisé selon les étapes suivantes :

- Élaboration de la base de connaissances
    - Base de cas ***légume***
    - Base de connaissances explicatives
- Élaboration du module de gestion de données pour la base de cas ***légume***
- Élaboration du module de RDP pour la classification
- Élaboration du module de gestion de données pour la base de connaissances explicatives
- Élaboration du module d'explication
    - Module d'aide
    - Explications de type quoi
    - Explications de type pourquoi
    - Explications de type comment
    - Explications de type pourquoi pas

Nous allons décrire dans ce qui suit les fonctionnalités et les interfaces du système. La figure 33 montre l'interface principale du système telle qu'elle apparaît lors du lancement de l'exécution.

VII.1.2 Le module de résolution de problème pour la classification

Ce module, dont l'interface est présentée en figure 33, permet de réaliser la classification de légumes potagers à l'aide de la fonctionnalité *Module de classification* qui permet d'accéder au module de classification dont l'interface est montrée en figure 34. Celui-ci permet, à partir de données entrées par l'utilisateur, par le biais d'une interface graphique, de la base de cas ***légume*** et du processus de résolution appliquant le raisonnement par cas pour un problème de classification, de retrouver la famille d'appartenance d'un légume dont la description n'est pas forcément complète. Le module RPC de classification a été développé avec Easy Reasoner, l'outil RPC de Eclipse. Il utilise deux algorithmes d'induction qui sont C4.5 et ID3. Dans notre application, ce sont les capacités en matière de classification de Easy Reasoner que nous avons utilisées. C'est la fonctionnalité *Classification* de l'interface de RDP (figure 34). Comme nous l'avons spécifié dans les modèles d'analyse et de conception, seul le processus RETROUVER a été implanté, les phases d'adaptation, de vérification et d'enregistrement étant laissées au soin de l'utilisateur qui peut les réaliser à l'aide du module de *gestion des cas* du module de RDP pour la classification (figure 34). En effet, la base de cas utilisée pour la classification est la base de cas ***légume***. La gestion des cas de cette base s'effectue directement depuis le module de résolution de problème. L'accès au module d'explication est rendu possible depuis le module de RDP.

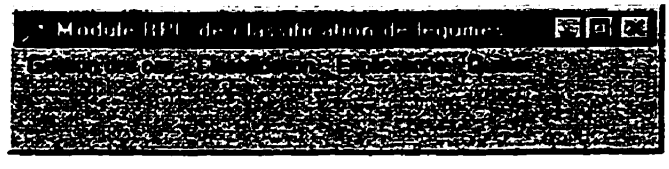

Figure 34. Interface pour le module de RDP pour la classification

L'interface de classification (figure 35) permet à l'utilisateur d'entrer les informations du légume considéré selon les descripteurs fournis. Il peut à tout moment demander des explications à l'aide de la touche F1 de son clavier ou en accédant directement au module d'explications par la fonctionnalité *explication*. Lorsque le cas est entré, il lance la recherche à l'aide de la fonctionnalité *classifier*. Il visualise les résultats de la classification dans la fenêtre prévue à cet effet. Les résultats sont représentés sous la forme suivante : la liste des familles de légumes pouvant classer le légume, et pour chacune le score obtenu après classification. L'utilisateur peut recommencer une autre recherche à l'aide de la fonctionnalité *Re*.

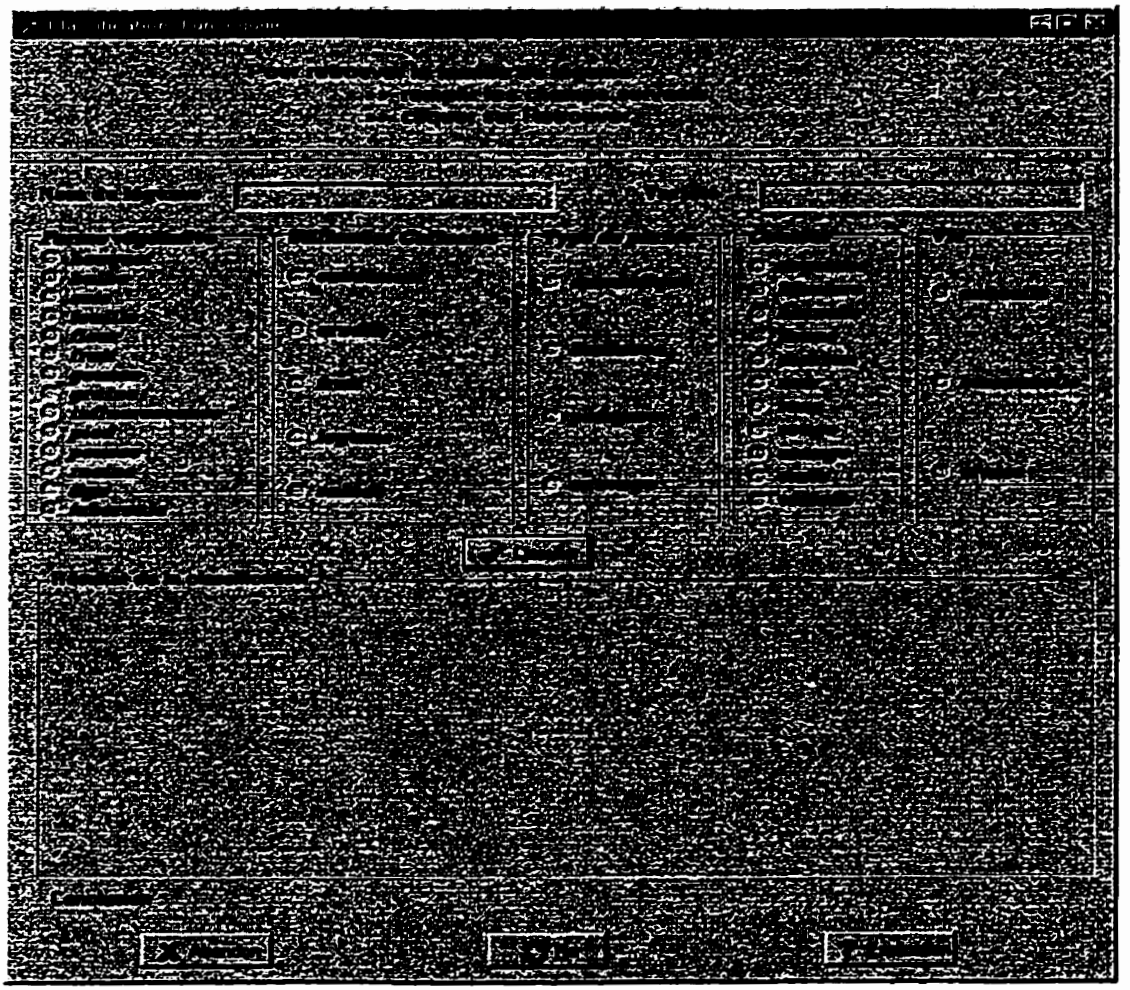

Figure 35. Interface pour la classification

VII.1.3 Le module de gestion des cas

Le module de gestion des cas permet d'accéder et de maintenir d'une part la base de cas de type ***légume*** telle qu'elle est utilisée par le module de RDP de classification et d'autre part la base de connaissances générale sur les légumes.

VII.1.3.1 Gestion de la base de cas de type *légume*

La description d'un cas de la base de cas de type ***légume*** propre à la classification est représentée en figure 36. Elle contient l'ensemble des descripteurs qui sont utilisés par le module de classification pour la résolution de problème. Le module de gestion permet de consulter, d'ajouter, de modifier et de supprimer les cas de cette base (figure 37).

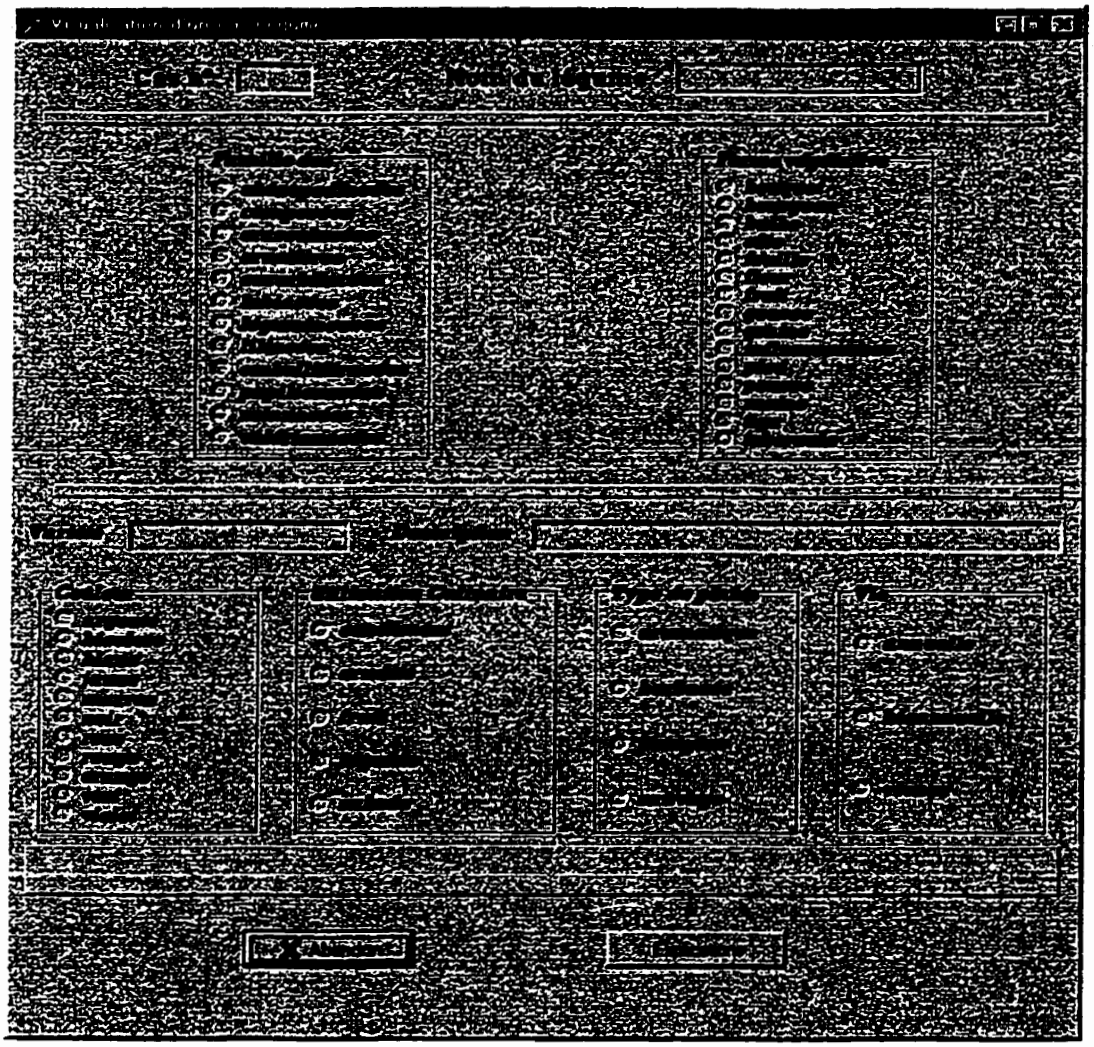

Figure 36. Interface d'un cas *légume*

Figure 37. Interface de Gestion des cas *légume*

VII.1.3.2 Gestion de la base de connaissances générale

La base de connaissances générale regroupe un ensemble de bases de données indexées qui sont connectées entre elles. Ainsi, la base de données principale est la base de données LÉGUME. Elle contient les descripteurs de la base de cas ***légume*** utilisée pour la classification, mais également d'autres descripteurs de légumes qui sont utilisés pour la configuration, mais aussi pour les explications. Ainsi, nous retrouvons dans la figure 38 les éléments qui composent la Base de Données LÉGUME. Il faut noter que les champs pour les données explicatives ne sont pas représentés ici. Il s'agit par exemple de la propriété image qui associe à un légume particulier une image. Cette propriété pourrait figurer sur cet écran. Ainsi, si nous n'avons pas implanté le module de configuration, l'acquisition des connaissances a été réalisée et l'information est stockée dans les bases de données correspondantes : relations d'amitié, de protection et d'ennemies. De plus les descripteurs de légumes utiles pour la planification tels que largeur, longueur, hauteur, etc. sont instanciés pour chaque légume de la base de données générale. Cette base de connaissances est accessible via l'interface principale du système par la fonctionnalité *base de cas*. Elle présente l'ensemble des légumes contenus dans la base, permet d'y naviguer, de visualiser, d'ajouter, de modifier et de supprimer des enregistrements de la base.

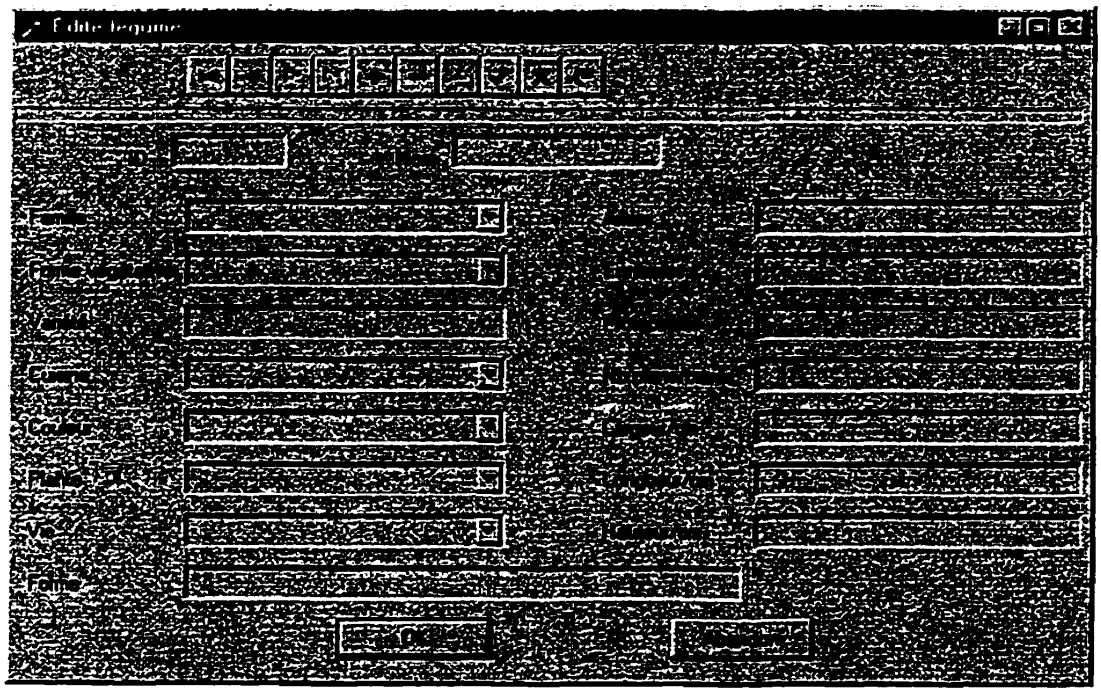

Figure 38. Interface Base de Connaissances LÉGUME

VII.1.4 Le module explicatif

Le module explicatif, ou module d'explications comme nous l'avons appelé, permet à l'utilisateur de pouvoir demander à tout moment des explications. Le type d'explication accessible à un instant donné est déterminé par l'étape de résolution de problème en cours. Ainsi, il permet, comme nous l'avons spécifié dans le modèle fonctionnel, de générer des explications sur les connaissances du domaine, mais aussi sur le raisonnement. L'accès au module peut être déclenché de deux manières : soit en utilisant la fonctionnalité *explication*, soit en utilisant le "clic souris droit". L'une ou l'autre des deux actions fait apparaître un menu qui permet d'accéder aux explications. Lorsqu'on se trouve dans l'interface principale, la fonctionnalité *module explicatif* affiche le menu du module (figure 39). La fonctionnalité *générateur d'explications* appelle l'interface indiquée sur la figure 40. Celle-ci permet de choisir un type d'utilisateur qui peut être *novice, technicien ou expert*. Le choix du niveau de complexité de l'explication n'est pris en compte à l'heure actuelle que pour les explications des connaissances qui répondent à la question quoi. Ensuite, l'utilisateur choisit le type d'explication qu'il désire entre *quoi, pourquoi, pourquoi pas, comment*. L'accès aux trois dernières fonctionnalités n'est rendu possible que si la RDP a été déclenchée, c'est-à-dire si un cas a déjà été présenté et résolu. Ainsi, l'accès au module d'explications depuis les autres interfaces déclenche l'interface présentée à la figure 41.

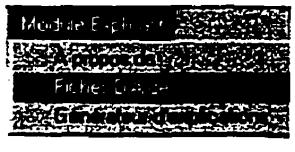

Figure 39. Interface du menu explication dans interface principale

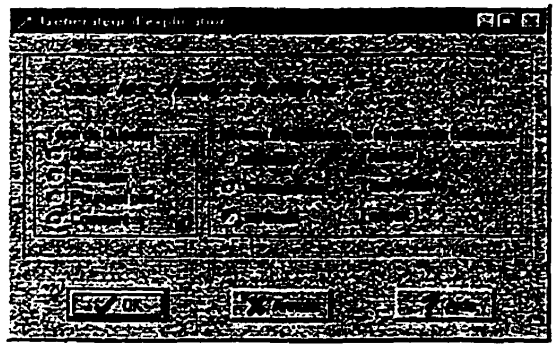

Figure 40. Interface du générateur d'explications

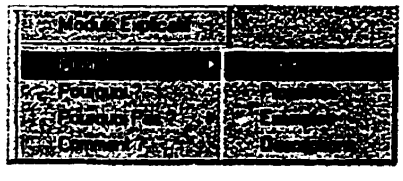

Figure 41. Interface du module d'explications de toute interface du système

VII.1.4.1 Les explications des connaissances

Les explications des connaissances sont de deux types. Elles correspondent à un module d'aide, et à des explications telles que modélisées pour la question de type quoi.

a) Module d'Aide

Comme dans tout logiciel actuel, l'existence d'un module d'aide accessible à l'utilisateur à tout moment représente une caractéristique fondamentale. Ainsi, même si cette propriété n'est en général pas énoncée dans les modules d'explications classiques, il doit en faire partie. En effet, l'utilisation d'un logiciel requiert de la part des utilisateurs une certaine accoutumance et une pratique, que ceux-ci ne possèdent en général pas dès le départ. De plus, si explication et aide ne peuvent être définis comme synonymes, la relation liant ces deux concepts est indéniable. C'est pourquoi, la mise en œuvre d'un module d'aide fait partie intégrante des explications concernant les connaissances. Ce module contient l'information qui permet :

- de renseigner l'utilisateur sur la page en cours de traitement, la méthode à suivre;
- dans le cas d'un logiciel de programmation comme par exemple Delphi ou Eclipse de définir la syntaxe des routines à utiliser, permet de trouver les noms des concepts à manipuler, de naviguer au travers du logiciel;
- de renseigner sur la résolution de problème utilisée : principes du cycle du RPC;
- de présenter le domaine d'application : principes du jardinage en carré, de la classification de légumes potagers et de la configuration de jardins.

Ce module a été développé avec l'outil spécifique de Windows pour la construction de modules d'aide : Microsoft Help Workshop.

L'aide qu'il fournit aux utilisateurs correspond à une aide statique. Les écrans d'aide correspondent à des fichiers textes dont le contenu a été défini. Cependant ceux-ci sont facilement modifiables en accédant directement aux fichiers. La figure 42 montre l'interface du *module d'Aide.*

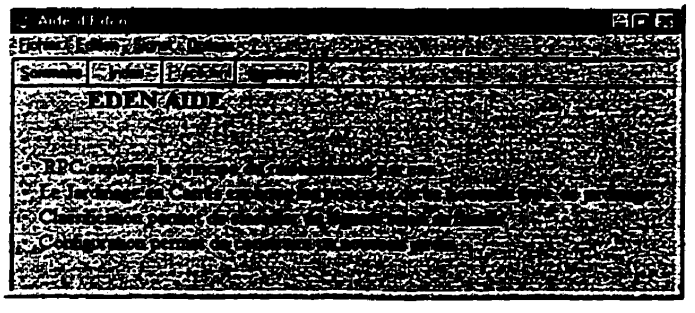

Figure 42. Interface du module d'aide

b) Définitions

C'est la première fonctionnalité du type d'explication quoi. Elle consiste à définir le terme entré par l'utilisateur dans le champ prévu à cet effet. Ce type d'explication prend en compte le niveau de l'utilisateur en donnant une définition plus ou moins complète. Ces explications ont été générées à l'aide d'Eclipse sous la forme de patrons d'explications statiques qui sont instanciés lors de l'appel de la fonction. C'est Delphi qui récupère l'explication produite pour la produire à l'écran. En effet, tout l'interfaçage est géré par Delphi. Ainsi on peut définir :

- *Ce qu'est une plante chénopodiacée*
- *Ce que sont deux plantes ennemies*
- *Ce qu'est un légume, etc...*

L'écran de saisie est montré en figure 43.

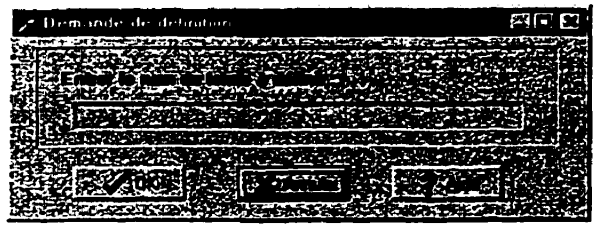

Figure 43. Interface de demande de définition

c) Exemples

La génération d'exemples permet à l'utilisateur de demander des exemples de légumes ou autres caractéristiques de légumes qui satisfont d'autres critères. Par exemple, cette fonctionnalité permet à l'utilisateur de demander :

- *Quels légumes appartiennent à la famille des chénopodiacées ?*
- *Quelles formes végétatives trouve-t-on dans la famille des solanacées ?*
- *Quels légumes ont la forme végétative tubercule ?*

Là, les explications ne correspondent pas à des patrons d'explications. Elles sont obtenues suite à un traitement de recherche dans la base de données en fonction des champs établis et sont gérées sous Delphi. Comme il s'agit d'exemples seuls les noms des termes retrouvés sont affichés. Un patron d'explication reprenant les données de problème peut être généré sous Eclipse. La figure 44 présente l'interface de cette fonctionnalité.

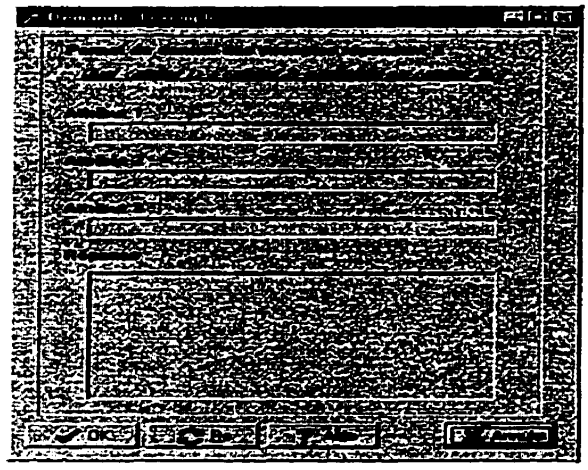

Figure 44. Interface de demande d'exemple

d) Propriété /Description

Ces fonctionnalités permettent d'accéder à une propriété spécifique d'un légume. Ces explications sont utiles pendant la résolution de problème pour comprendre la phase de décision lors de la construction de l'arbre de décision pour la classification. Ces explications ont été développées sous Eclipse à l'aide de patrons d'explications statiques. Elles correspondent aux question du type :

- *Quelles sont les propriétés potagères de la carotte ?*

La figure 45 montre l'interface de cette fonctionnalité.

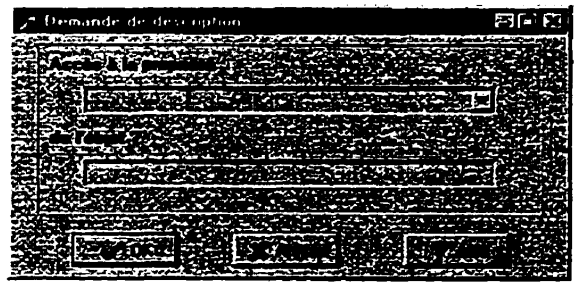

Figure 45. Interface de demande de propriété

Finalement l'ensemble des explications statiques d'EDEN est généré à partir de patrons d'explications statiques instanciés lors de l'appel, facilement accessibles et modifiables et dont le résultat repose sur la bonne gestion des bases de données qui contiennent les informations. Toutes les connaissances ne sont toutefois pas disponibles dans les bases de données. Certaines d'entre elles, notamment pour les définitions, sont directement implantées sous Eclipse sous la forme de faits et règles.

VII.1.4.2 Les explications du raisonnement

Nous venons de présenter les fonctionnalités qui permettent d'élaborer des explications statiques. Nous allons maintenant décrire celles qui constituent les explications dynamiques d'EDEN. Nous avons défini dans la phase de conception qu'il existe trois types d'explications du raisonnement : pourquoi, pourquoi pas, comment. Néanmoins, nous devons noter que la génération des types suit un ordre établi. Le premier type est le type *comment*. Chacune des explications sera illustrée par un exemple dans la section VII.2.

a) Comment ?

Comme nous l'avons mentionné plus tôt, le processus de RDP de classification a été implanté avec l'outil Easy Reasoner. Le principe de classification RPC d'EDEN repose sur la construction de l'arbre d'indexation tel qu'il a été créé par Easy Reasoner pour pouvoir effectuer la classification des légumes potagers en fonction de leur famille d'appartenance. Easy Reasoner se comporte comme une coquille RPC, à savoir que la méthode utilisée n'est pas accessible au concepteur. C'est une sorte de "boîte noire", dans laquelle on met les paramètres d'entrée et qui fournit les résultats obtenus sous la forme de probabilités. Aussi, pour élaborer des explications sur comment fonctionne le processus de résolution, nous avons dû utiliser directement la trace du raisonnement. La construction d'un arbre d'indexation avec Easy Reasoner se résume à l'utilisation de quatre fonctions. La première (CBR BUILD DECISION...) permet de définir le domaine en déterminant les données utilisées pour la classification, de définir l'objet de la classification c'est-à-dire le champ à prédire et les champs entrant en jeu pour réaliser la prédiction, ensuite, de déterminer l'algorithme de recherche utilisé, à savoir par la méthode du plus proche voisin ou par arbre de décision. La deuxième (CBR ASSERT INDEX ...) initialise l'arbre d'indexation. La troisième (CBR ASSERT FIELDS ...) initialise les nœuds de l'arbre d'indexation. Finalement, la quatrième (CBR ASSERT TREE) rattache les nœuds à l'arbre. Aussi, est-il difficile de produire des explications à partir du faible nombre d'informations à notre disposition, les traitements relatifs à chacune de ces fonctions n'étant pas accessibles. L'ordre de déclenchement des règles énoncées plus tôt est la seule source d'information existante. Ceci dit, la consultation d'un ouvrage décrivant les méthodes d'indexation utilisées par Easy Reasoner permet d'initialiser une fiche d'aide avec les principes de ces méthodes pour des utilisateurs intéressés à connaître les algorithmes d'induction comme ID3 et C4.5. Néanmoins,

Easy Reasoner n'est pas à ce point fermé que l'on ne puisse y trouver des informations utilisables pour générer des explications utiles à tout utilisateur. Ainsi, si l'on ne peut expliquer la méthode propre de construction de l'arbre , on peut représenter l'arbre tel qu'il a été créé avec pour chaque nœud les informations pertinentes permettant à l'utilisateur de comprendre la structure de l'arbre, les éléments le composant, et ainsi expliquer à l'utilisateur l'organisation des données dans la base de cas constituée par Easy Reasoner. Ainsi, la réponse à la question ***Comment*** ? concerne la méthode utilisée lors de la résolution de problème à savoir l'ordre de déclenchement des règles, les faits en entrée et les résultats en sortie. Ce type d'explication ne prend pas en compte le cas traité, il ne fait que montrer la trace de résolution qui a permis de construire l'arbre de décision. C'est pourquoi nous avons choisi de représenter les explications à l'aide d'un arbre binaire dont chaque nœud permet de discriminer des classes de légumes. Ces explications permettent donc de comprendre la méthode suivie par Easy Reasoner. Elles ont été élaborées selon la méthode suivante: récupération de la trace du raisonnement d'Éclipse qui contient les faits initiaux, les règles déclenchées et les faits déduits. Seuls les éléments pertinents et utiles à la génération d'explications sont conservés de la trace initiale suite à un filtrage. Ensuite, la trace est fournie à Delphi qui structure les données recueillies pour représenter la construction de l'arbre de décision. Ainsi, un historique est créé conservant toutes les données qui ont permis d'élaborer l'arbre et à partir de quelles valeurs. L'arbre correspond à une liste d'objets qui contiennent la description et l'état de chaque nœud.

b) Pourquoi ?

Cette fonctionnalité permet d'expliquer le résultat trouvé pour le cas traité. Il réalise la trace du raisonnement en reprenant la trace élaborée pour répondre à la question comment mais en n'explorant que les branches qui ont conduit au résultat. Ceci permet de montrer à l'utilisateur pourquoi le cas légume considéré a été classifié dans une famille et pourquoi il a obtenu le score. De plus, cette fonctionnalité offre une justification des résultats obtenus en expliquant les calculs effectués par Easy Reasoner. Elle montre aussi l'ordre de déclenchement des règles en fonction des valeurs traitées par rapport au nœud donné. Ceci permet de donner une explication pas-à-pas du raisonnement suivi lors de la résolution du cas considéré.

c) Pourquoi Pas ?

Cette dernière fonctionnalité a pour rôle de montrer à l'utilisateur la pertinence des résultats trouvés. Elle lui demande quel résultat il aurait voulu trouver et lui montre le parcours dans l'arbre qui aurait mené à celui-ci ainsi que la différence et les erreurs produites par rapport au parcours exact. Il permet pour chaque nœud de visualiser la liste des valeurs attendues et la valeur obtenue. En montrant les deux chemins de résolution, l'utilisateur en comparant les règles et les valeurs associées à chacun peut ainsi comprendre son erreur.

Les explications du raisonnement reposent donc sur la trace obtenue suite à la résolution de problème réalisée par Easy Reasoner sur un cas traité et sont construites et gérées par Delphi qui organise et structure les données recueillies pour qu'elles aient un sens. Les explications apparaissent sous la forme d'un arbre commenté et dont chaque nœud est expliqué. Nous montrerons les écrans manquants dans la section suivante qui consiste à présenter EDEN sur un exemple concret.

VII.2 Illustration sur un exemple

Nous avons décrit dans le paragraphe précédent les fonctionnalités du prototype EDEN. Afin d'illustrer les capacités explicatives de notre application, nous allons présenter un exemple et montrer les étapes du processus de résolution.

Pour commencer l'exécution d'EDEN il est nécessaire de lancer l'exécutable *EDEN.exe*. L'interface principale de l'application apparaît telle qu'elle a été montrée en figure 33. L'utilisateur qui n'est pas familier avec le domaine peut lancer le module d'aide de la fonctionnalité *Module explicatif*. Ensuite, il lance le module de résolution de problème en cliquant sur *Module de Classification* et de celui-ci actionne l'option *Classification* du menu (figure 34). L'écran de la figure 35 apparaît. Afin de remplir les champs qui sont demandés, l'utilisateur peut faire appel au module d'explications. Par exemple, s'il ne connaît pas la signification du champ *variété*, il peut demander la définition de ce terme ou s'il veut connaître les éléments qui caractérisent une carotte par exemple. Le résultat donné est montré en figure 46.

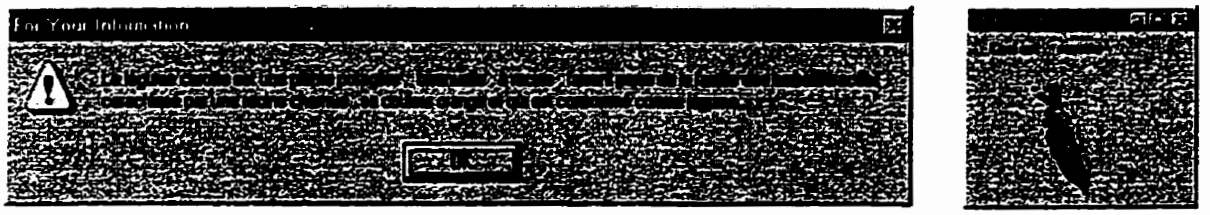

Figure 46. Définition d'une carotte

Quand l'utilisateur a rempli les champs qui l'intéressent, notons que EDEN supporte les valeurs manquantes, l'utilisateur lance la recherche. Le résultat obtenu est montré en figure 47.

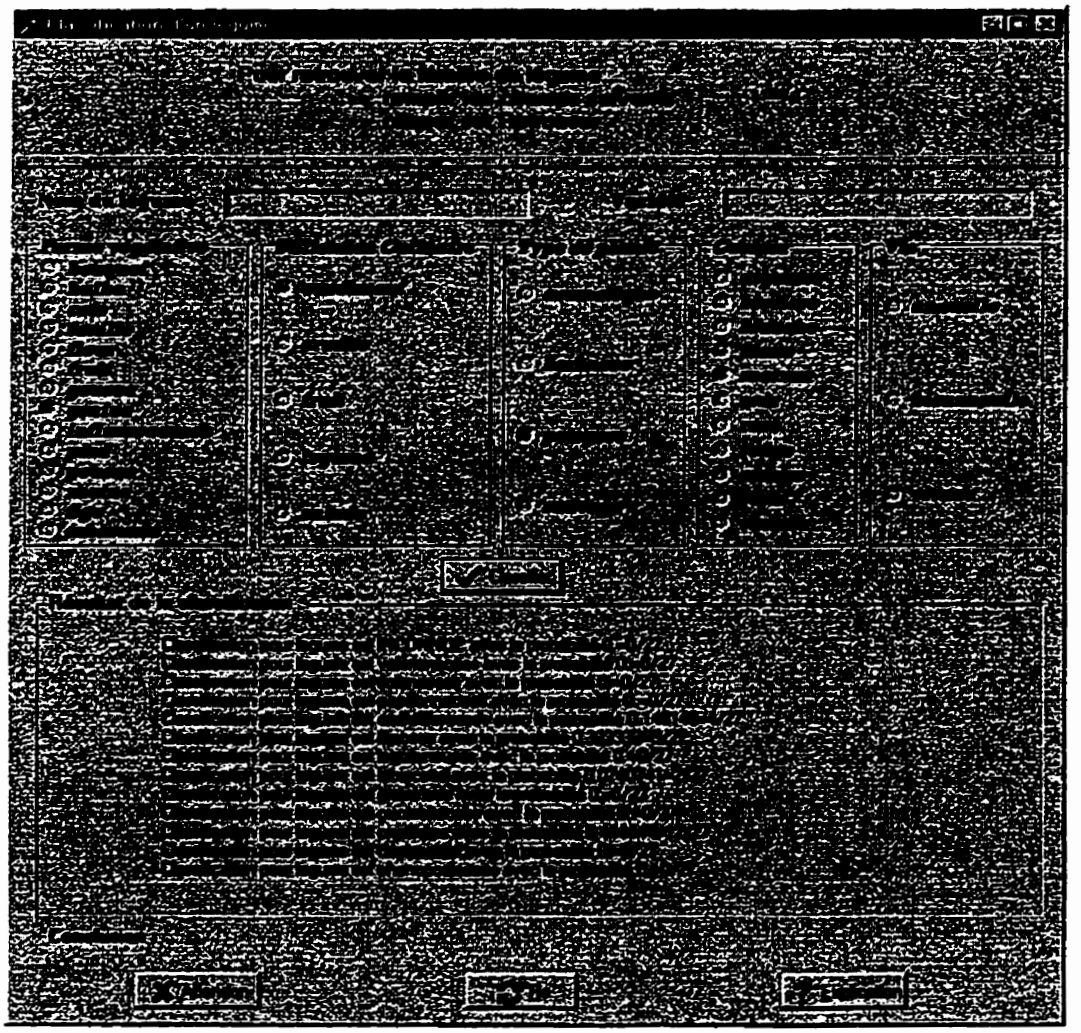

Figure 47. Résultat de la classification

Ainsi dans notre exemple, l'utilisateur a voulu trouver la famille d'appartenance des légumes dont :

- la forme végétative = graine,
- l'utilisation culinaire = condiment,
- type de plante = potagère
- la couleur = marron.

Nous noterons ici que le fait de donner le nom du légume n'est pas très intéressant pour classifier un légume, mais qu'il peut être donné pour évaluer Easy Reasoner et vérifier qu'il retrouve bien la famille du légume.

La figure 47 montre le résultat obtenu par Easy Reasoner trouve. Il est évident qu'à moins d'être expert et d'avoir un ordinateur soi-même dans la tête, le résultat n'est, à première vue, pas significatif et encore moins compréhensible. C'est pourquoi, même en tant que concepteur, nous avions besoin d'explications pour comprendre le fonctionnement même du logiciel utilisé. Mais plaçons-nous ici en tant que simple utilisateur. Donc la grille des résultats nous apprend que le légume que nous avons décrit a 4 chances sur 27 d'appartenir à la famille des ombelliféracées, 9/27 aux liliacées, 2/27 aux labiacées et 12/27 aux crucifères. La première question pourrait être de se demander ce qui différencie la famille des ombelliféracées de celle des labiacées. Pour cela, l'utilisateur peut faire une demande de définition. La figure 48 montre les définitions obtenues.

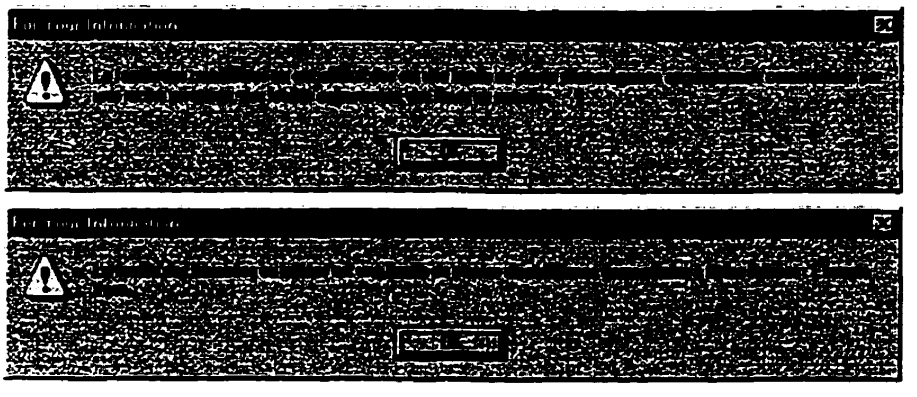

Figure 48. Définitions des termes : ombelliféracées et labiacées

Déjà l'utilisateur en connaît davantage sur la terminologie. Il peut continuer ainsi à demander d'autres définitions. Cependant, il n'a toujours aucune information lui expliquant ce que ces résultats peuvent bien vouloir signifier. C'est pourquoi il active la propriété *pourquoi* du menu *explication*. L'écran de la figure 49 apparaît. EDEN montre la représentation graphique de l'arbre de résolution correspondant au cas traité.

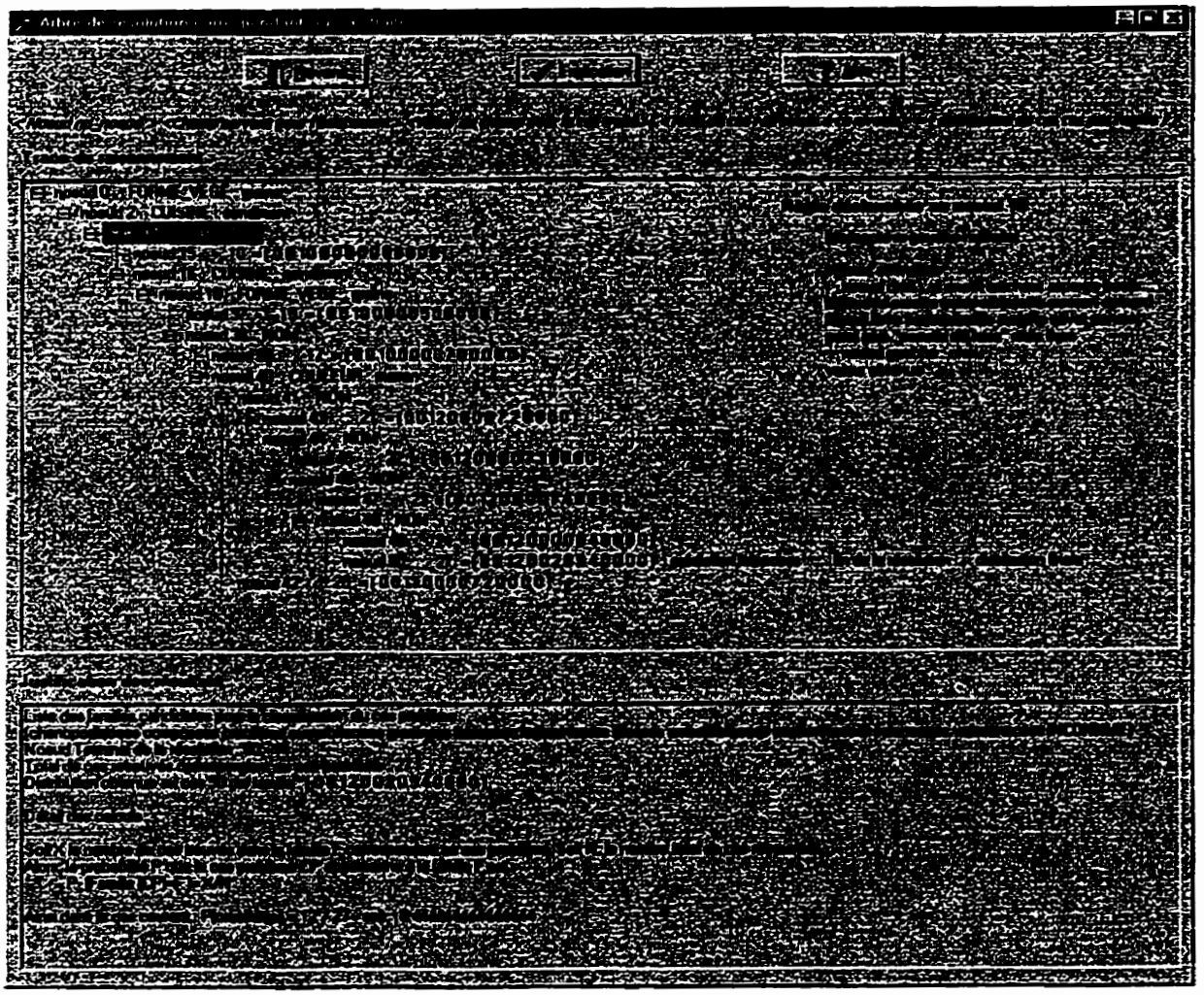

Figure 49. Pourquoi ce résultat ?

Plusieurs éléments sont présents sur cette interface. Tout d'abord, des boutons de contrôle, comme dans toutes les fiches d'EDEN, permettent de naviguer entre les écrans du système. Ensuite, une fenêtre contient la trace du raisonnement. Sur chaque ligne il y a : le nœud de l'arbre avec son numéro, le premier nœud est le nœud 0; le nom du champ qui a permis de sélectionner l'une des deux branches de l'arbre et de choisir le nœud fils prochain qui sera inspecté; la valeur du champ pour le cas traité; le total des cas qui sont à ce niveau de la

décomposition et finalement la distribution des cas à ce nœud par famille. La justification des résultats explique à quoi correspond la distribution, quel est le nœud terminal de l'arbre de recherche, le nombre de cas retrouvés avec leur distribution par famille et, finalement, l'explication des résultats numériques obtenus pour chaque famille. L'utilisateur peut en cliquant sur les nœuds de l'arbre voir quelle règle a été déclenchée pour l'éclatement du nœud, les valeurs attendues sur chaque nœud enfant et la valeur correspondante du cas traité. Ceci permet d'expliquer pourquoi une branche a été développée et pas l'autre. Les règles déclenchées sont les suivantes :

- *(descente ni gauche ni droite)* : cette règle indique que la valeur du champ pour le cas traité ne correspond à aucune des valeurs attendues ou n'a pas été identifiée. Aussi, les deux branches de l'arbre sont explorées. Exemple : le nœud 10 qui n'a pas de valeur pour le champ VARIÉTÉ.
- *(descente symbole droit)* : indique que la branche droite de l'arbre est explorée car la valeur du champ demandé correspond à celle donnée dans le cas. Exemple: au nœud 0, la valeur graine du champ FORME-VEGE permet de développer le nœud 2.
- *(descente symbole gauche)* : identique à (descente symbole droit) mais fait la recherche dans le sous-arbre gauche. Exemple : il n'y en a pas pour le cas traité.
- *(initialisation distribution)* : lorsque le premier nœud de l'arbre est un nœud terminal, la distribution par famille des cas présents à ce nœud terminal est calculée.
- *(mise à jour distribution)* : la distribution précédemment calculée est mise à jour en ajoutant la distribution des cas qui se trouvent au nœud terminal courant.

Ainsi la distribution finale contient tous les cas qui ont été retrouvés lors du parcours de l'arbre et leur appartenance à la famille. Le parcours de cet arbre permet de faire le choix dans la classification finale que l'on attribuera au cas traité. En effet, considérons le cas du nœud 40 (figure 50). On constate alors que la recherche dans l'arbre s'est poursuivie alors qu'aucune des valeurs attendues (orange, rose, blanche, verte) ne correspondait à la valeur obtenue (marron). La recherche aurait donc dû s'arrêter là, puisque les cas retrouvés ensuite ne correspondent pas aux valeurs initialement entrées. Ceci permet à l'utilisateur de conserver la dernière distribution obtenue au nœud 39 et de l'utiliser pour choisir la famille qu'il attribuera au légume qu'il considère. Nous venons donc de décrire la fonctionnalité *pourquoi* du module explicatif d'EDEN.

Figure 50. Problème de désaccord rencontré à un noeud

L'utilisateur même s'il a compris pourquoi le cas qu'il considère a été classifié selon la distribution donnée ne sait toujours rien concernant la méthode qui a été suivie pour construire l'arbre de décision. Il peut alors actionner la fonctionnalité *comment* du module d'explication. Celle-ci l'amène à une fenêtre (figure 51) qui contient elle aussi un graphe représentant la décomposition d'un arbre. L'arbre représenté contient l'arbre d'indexation tel qu'il a été créé par Easy Reasoner à partir des données en sa possession à savoir la base de cas, le champ à prédire et la liste des champs servant pour la prédiction. Notons ici que les champs qui ont été considérés ici pour prédire la classe d'un légume sont : le nom, la forme végétative, la variété, l'utilisation culinaire et la couleur et que modifier cette liste est très facilement réalisable.

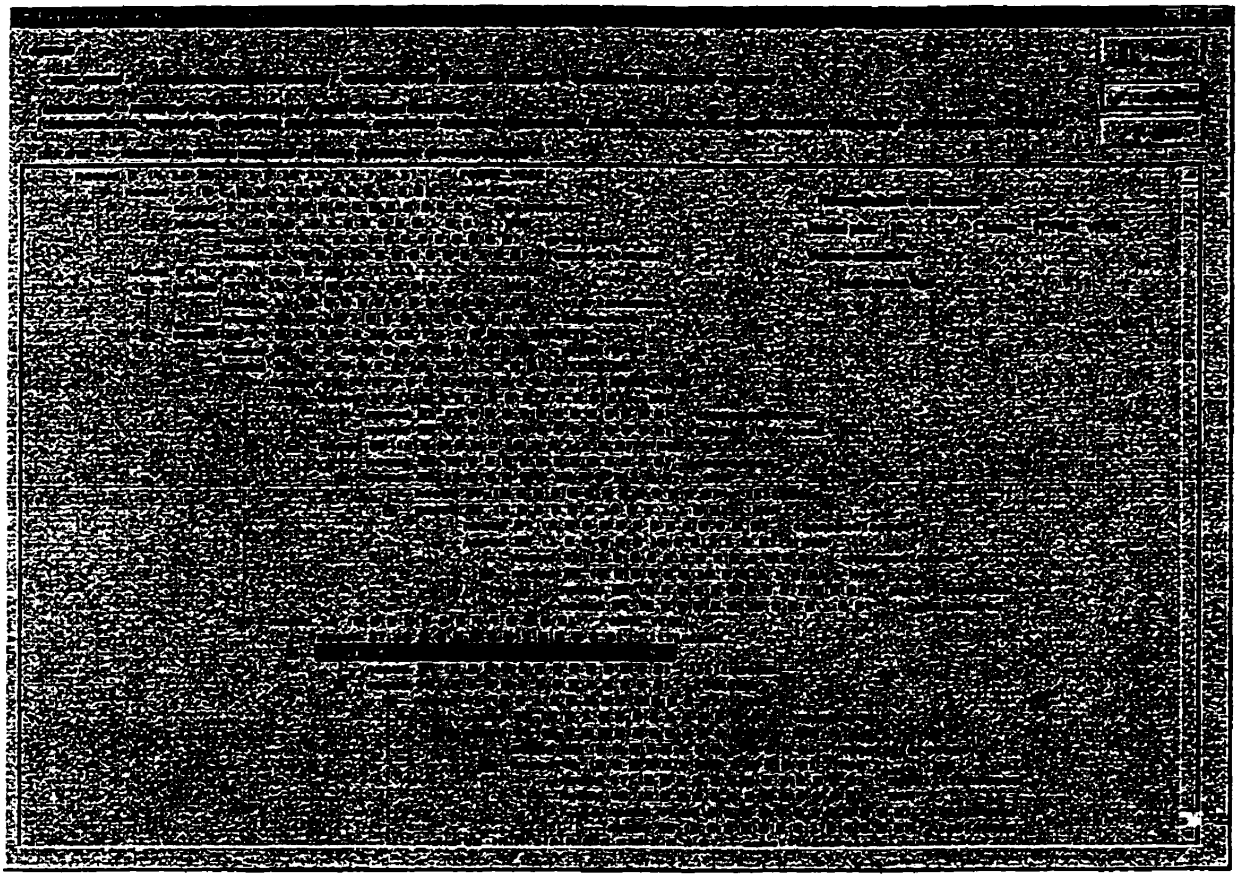

Figure 51. Comment l'arbre a-t-il été créé ?

L'arbre d'indexation est donc recréé à chaque fois que la base de cas est mise à jour par l'ajout, la modification ou la suppression d'un cas ou que la liste des cas servant pour la classification est changée. Les données contenues dans l'arbre sont les seules qui ont pu être récupérées. Elles permettent d'évoluer dans l'arbre et de voir la situation à chaque nœud. Ainsi, au nœud 38, on apprend que le nœud père est le nœud 18, que le champ du nœud père qui a été utilisé pour l'éclater était le champ forme végétative et que les valeurs de ce champ au nœud 38 sont feuille, racine et tige, enfin on voit sur la ligne du nœud 38 la distribution des cas par familles à ce nœud et le champ qui va être considéré pour le nœud suivant. On peut déduire de l'observation de l'arbre que le choix d'un champ d'un nœud à l'autre est fait pour discriminer les familles au maximum. Les règles qui ont été utilisées par Easy Reasoner pour créer l'arbre d'indexation n'étant pas disponibles c'est pourquoi elles ne figurent pas dans l'interface. Néanmoins, malgré le peu d'informations qui ont pu être acquises, l'arbre d'indexation constitue une bonne source d'information pour comprendre comment un cas est classifié. En effet, chaque nœud terminal montre la famille qui est déduite à ce nœud de l'arbre. Aussi le

parcours de l'arbre permet d'expliquer comment arriver à classifier une famille. Cette constatation nous permet de considérer la dernière option d'explication qui peut être générée dans EDEN à savoir répondre à la question *pourquoi pas*.

Ainsi, revenons à l'écran montrant les résultats de la recherche. L'utilisateur, s'il connaît un peu la classification des légumes, peut paraître surpris car les classes de légumes qu'il attendait n'ont obtenu que des probabilités 0. Il va alors faire appel au générateur d'explications des connaissances pour vérifier si ses propres croyances sont correctes. Par exemple, il lui semblait que les légumes dont la forme végétative est une graine appartiennent à la famille des légumineuses. Il veut donc confirmer son résultat. Pour cela il utilise la fonctionnalité *exemple* du module *quoi* dans laquelle il demande *quelles familles ont des légumes qui ont une forme végétative légume ?* Le résultat de la recherche est montré dans la figure 52.

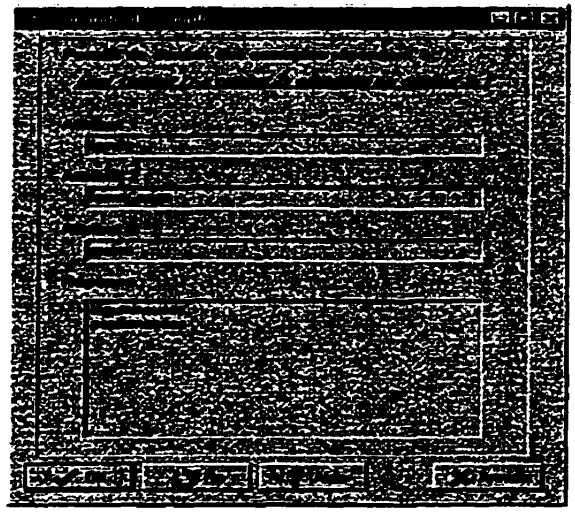

Figure 52 Demande de confirmation par un exemple

Fort de ces connaissances il va alors demander au système pourquoi la famille des légumineuses qui pourtant lui semblait correcte n'a pas été retrouvée en choisissant l'option *pourquoi pas* du menu des explications. Le résultat obtenu est montré à la figure 53. Nous constatons alors que l'arbre qui est représenté ici montre les chemins de l'arbre qui mènent à la classe voulue par l'utilisateur. Le principe de cet arbre consiste à comparer les valeurs attendues pour classer le cas dans la classe voulue et les valeurs trouvées du cas. Ainsi, au nœud 17 on constate que pour que le cas traité soit classé dans la classe des légumineuses il aurait fallu que sa valeur pour le champ cuisine soit légume, or celle-ci est un condiment et tous les légumes de la classe des légumineuses ont une utilisation culinaire en tant que

légume. Ceci permet à l'utilisateur d'apprendre de ses erreurs et de devenir ainsi plus compétent en matière de classification de légumes potagers.

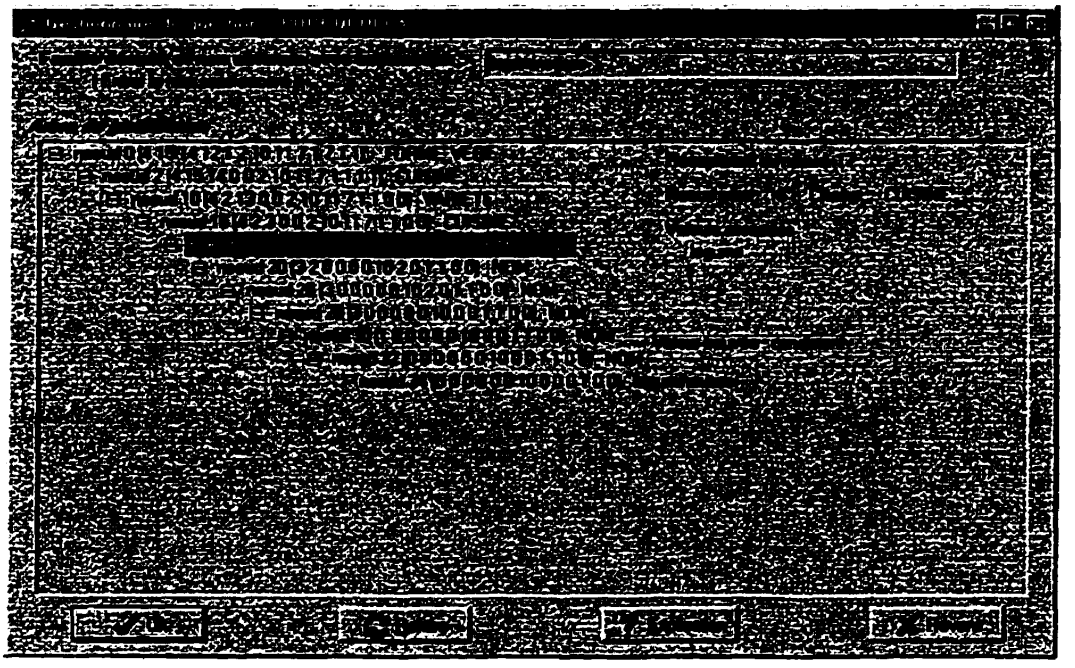

Figure 53. Pourquoi la classe des légumineuses n'a-t-elle pas été retrouvée ?

Nous venons donc de décrire par un exemple l'ensemble des fonctionnalités explicatives d'EDEN. Nous n'avons pas détaillé les phases d'adaptation et d'enregistrement car, comme nous l'avons spécifié dans la phase de conception, celles-ci sont laissées aux soins de l'utilisateur. À force d'accès au module explicatif lui donnant les définitions et les justifications qu'il désire, celui-ci pourra compléter les champs qui lui manquent pour la description du légume, choisir la solution qui lui semble la plus pertinente suite à l'étude de l'arbre et insérer le nouveau cas résolu dans la base de légume à l'aide du module de gestion. La résolution suivante prendra en compte le nouveau cas résolu pour créer un nouvel arbre d'indexation qui sera utilisé pour les recherches suivantes. Nous en avons donc terminé ici avec la description du prototype EDEN, de ses fonctionnalités et capacités explicatives, qui sont le résultat de l'implémentation de l'application que nous avons définie tout au long de ce mémoire. Aussi, nous allons clôturer ce chapitre en faisant une discussion sur les résultats que nous avons obtenus et sur notre travail en général.

VII.3 Discussion

Le prototype EDEN a été développé afin de montrer l'applicabilité dans un SBC-RPC du modèle générique d'explications que nous avons élaboré. Ceci constituait l'un des objectifs de notre recherche. En effet, le but initial poursuivi consistait à développer un modèle qui permette d'avoir une vue générale sur le domaine particulier de la génération d'explications dans des SBC et plus précisément dans des SBC-RPC. Les résultats obtenus ne sont certes pas complets, mais illustrent de manière significative les capacités explicatives que de tels systèmes peuvent avoir. Nous avons en effet implanté l'ensemble des types d'explications que nous avions définis sur les connaissances et le raisonnement. De plus nous avons montré l'importance de chacun des critères contenus dans le modèle générique et leur rôle pendant l'application de la méthode KADS pour modéliser notre système. Le résultat de la modélisation, à savoir l'organisation et l'analyse des connaissances utiles pour les explications dans un contexte de RDP particulier, nous a confirmé notre choix pour la méthodologie de développement KADS. En effet, la génération d'explications nécessite une modélisation explicite des connaissances à un haut niveau d'abstraction pour générer des explications dynamiques en fonction du niveau de complexité et d'abstraction des données du problème. C'est pourquoi la méthode KADS est adaptée, la modélisation au niveau des connaissances étant sa principale caractéristique. Aussi, nous nous sommes inspirée des travaux de plusieurs chercheurs dont [Bourcier *et al.* 94; Leake 95; Sprenger 93] pour commencer nos travaux sur la typologie des explications. Néanmoins nous n'avons pas, comme [Sprenger 93; Baumewerd-Ahlmann *et al.* 90], utilisé la méthode KADS pour étudier à quels niveaux du modèle d'expertise de la résolution de problème les connaissances explicatives, en fonction des types de questions posées, devaient être analysées, mais pour modéliser la génération des explications même. De plus, nous sommes partie du principe, comme [Bourcier *et al.* 94; Martin 94], que la génération d'explications est une tâche de résolution à part entière qui nécessite ses propres techniques de résolution, c'est pourquoi nous avons, à l'idée de Martin, réalisé un modèle d'expertise spécifique aux explications dans les SBC-RPC et qui permette une réutilisabilité facile donc à un assez haut niveau d'abstraction. De plus, nous n'avons pas produit de modèle d'expertise pour la communication puisqu'il n'était pas utile pour notre application. Finalement, nous avons trouvé intéressante l'idée de Bourcier [Bourcier *et al.* 94] de considérer un modèle spécifique pour le cas traité, car elle est directement appliquée dans le cadre du RPC qui n'utilise que des cas et dont la résolution part d'un cas problème, aussi

contrairement à ses travaux, aucun effort n'est requis pour considérer le cas traité. KADS nous a donc permis de réaliser une modélisation de la génération d'explications qui peut être facilement adaptable et applicable dans tout autre SBC-RPC qui aurait les mêmes besoins en explications que ceux considérés dans le modèle, c'est-à-dire utiliser les explications comme un outil d'apprentissage orienté utilisateur. En effet, nous avions identifié trois catégories de systèmes de RPC utilisant les explications (section III.1). La première concerne les systèmes pour lesquels les explications représentent un outil d'apprentissage permettant au système de réparer les erreurs commises, ceux-ci sont traités notamment par [Leake 95; Schank *et al.* 94]; le domaine d'application choisi ne s'appliquait pas à cette optique. La deuxième utilise les explications comme outil d'aide pour améliorer les performances du système et font l'objet des travaux de plusieurs chercheurs dont [Bento *et al.* 94a; López de Mántaraz&Plaza 97]. L'approche que nous avons suivie et qui correspond à l'objectif fixé s'applique aux première et quatrième catégorie, à savoir considérer les explications comme un outil destiné à expliquer à l'utilisateur, et non pas au système, les connaissances manipulées et le raisonnement suivi et cela, en tant qu'outil d'apprentissage également. Notre objectif est atteint dans ce sens que non seulement le prototype EDEN fournit des explications des connaissances, par exemple sous forme d'écrans d'aide standards, de patrons d'explications ou d'images, mais explique aussi le raisonnement. L'outil que nous avons utilisé pour implémenter le module de classification de RPC d'EDEN, Easy Reasoner, s'est en fait avéré être un environnement fermé. Nous en connaissions les fonctionnalités,. mais ne l'avions pas utilisé auparavant. Ceci nous a permis de montrer que des explications dynamiques du raisonnement peuvent être produites même si l'outil utilisé ne permet pas de récupérer les connaissances nécessaires pour expliquer son raisonnement. De plus, cela nous a permis d'explorer un autre type de catégorie d'explication proche de la deuxième citée plus haut. En effet, le fait d'élaborer des explications sur le raisonnement suivi par Easy Reasoner pour accomplir la tâche d'explication nous a permis de comprendre la méthode suivie par le logiciel alors que celui-ci agissait comme une boîte noire avec en entrée un ensemble de données et un résultat en sortie. De plus, les explications nous ont permis de critiquer les résultats obtenus et s'avèrent être un bon moyen d'apprentissage non seulement pour le domaine d'application mais aussi pour le raisonnement suivi. Nous avons pu constater que l'utilisateur pouvait être en désaccord avec les résultats obtenus et ainsi d'élaborer ses propres logique ou raisonnement pour le futur. Évidemment le fait de n'avoir pas implanté le

module de configuration ne nous a pas permis d'explorer et de confirmer notre hypothèse selon laquelle les explications ont un rôle très important pendant les phases d'adaptation et de validation du RPC et qu'elles sont appropriées pour des tâches de synthèse comme la planification qui nécessite des traitements plus complexes que la classification. Cependant nous pensons que la représentation des explications sous forme d'arbre présente un certain nombre d'avantages. Tout d'abord la représentation graphique d'un arbre avec, pour chaque nœud, le détail des informations explicatives de l'étape en cours, est facilement accessible et donc compréhensible à tout utilisateur. Cela permet d'avoir une vue d'ensemble du raisonnement suivi sous une forme condensée. De plus, la recherche d'information dans cet arbre est plus rapide et efficace. Du point de vue de la validité de la méthode, le formalisme de représentation utilisé offre une maintenance facile. La structure de l'arbre consiste en une liste de nœuds facilement modifiables. De plus elle est réutilisable; en effet, pour tout système utilisant une méthode d'induction, de recherche dans un arbre indexé ou non, la méthode est applicable. Ainsi pour le module de planification un arbre de recherche est aussi utilisé. Il ne s'agit pas d'un arbre binaire et les contraintes de recherche sont plus nombreuses et complexes que pour la classification, de plus, la recherche dans l'arbre est une recherche en profondeur d'abord, puis en largeur ensuite. En effet, à la différence de la classification telle que réalisée par Easy Reasoner l'exploration d'une branche d'un arbre échoue lorsqu'un fait n'est pas vérifié. Ce sont les explications générées par EDEN qui nous ont appris cette erreur dans le raisonnement suivi par Easy Reasoner. La réutilisabilité pourrait également être effectuée dans d'autres applications, par exemple pour un prototype de RPC sur la sécurité routière que nous avons développé [Verrière&Tourigny 98; Capus&Tourigny 98a]. Celui-ci utilise également des techniques de recherche dans un arbre indexé. La similarité est alors facilement explicable en étudiant le parcours dans l'arbre ainsi que toutes les caractéristiques du RPC. Nous avons depuis le début de ce paragraphe énuméré de manière non formelle l'ensemble des qualités des explications d'EDEN. Celles-ci répondent aux critères d'évaluation des explications que nous avons définies dans le chapitre II.1.4.2. Aussi malgré le fait que toute l'implémentation n'a pas été faite, nous pouvons considérer que le modèle que nous avons développé est valable ainsi que l'approche que nous avons suivie pour le réaliser. Les résultats de notre travail ayant été évalués, il ne nous reste plus qu'à finir notre mémoire en faisant la conclusion de cette recherche.

CONCLUSION

L'objectif général de ce mémoire était de proposer une méthode pour la génération d'explications dans des SBC-RPC. Celle-ci est décrite au travers des chapitres IV, V, VI et VII. En effet, nous avons d'abord élaboré un modèle générique d'explications en faisant l'inventaire à partir d'une étude bibliographique des critères à prendre en compte pour générer des explications dans des SBC-RPC. Nous avons orienté nos recherches en essayant de déterminer quels sont les besoins des utilisateurs qui se servent d'un SBC-RPC. Le modèle construit a ensuite été développé en utilisant la méthode KADS qui représente un outil adéquat pour effectuer l'analyse et la conception d'un module d'explications en accord avec les critères définis dans le modèle générique. L'applicabilité et la validité de la méthode ont été démontrées avec l'implémentation du prototype EDEN qui est une application de SBC-RPC dans le domaine de la classification de légumes potagers et de la planification de jardin en carrés. Ce travail nous a permis de définir une méthode qui est suffisamment générale pour être réutilisée dans d'autres applications de SBC-RPC pour élaborer des dispositifs explicatifs. De plus, le modèle générique d'explications que nous avons construit permet de déterminer dès le début de l'élaboration d'un tel dispositif les éléments importants, et la méthode KADS permet de les organiser et de modéliser les connaissances utiles à la génération d'explications au bon niveau. Finalement, nous avons pu traiter la génération d'explications dans les SBC-RPC, à l'aide d'une méthode bien définie, dans un contexte qui permet aux utilisateurs de mieux comprendre le raisonnement utilisé, le domaine considéré et les connaissances manipulées, dans une même application, à l'aide du prototype EDEN, alors que les travaux réalisés ne considèrent généralement qu'une de ces particularités.

Toutefois, il reste encore quelques points qui mériteraient d'être réalisés dans le futur pour compléter ce travail:

- Application du modèle à d'autres types de tâches expertes, par exemple la planification, en intégrant le module de planification de jardin en carrés au prototype EDEN.
- Capacité de générer des explications en langage naturel. Par exemple, permettre à un utilisateur de formuler ses questions en langage naturel et concevoir un module pour les interpréter et générer des explications appropriées. Cette éventualité a été considérée dans le module d'expertise pour les explications du chapitre VI.
- Application de la méthode dans une application SBC-RPC réelle pour tester la généricité et la validité et enrichir le modèle générique proposé avec de nouvelles stratégies explicatives. En effet, l'application de jardinage en carrés a été conçue pour exploiter les types d'explications et les critères définis dans le modèle générique.
- Tester l'utilité des explications générées avec l'implémentation de la méthode auprès d'organismes. Il s'agirait de montrer notre prototype EDEN à des organismes soit liés à l'éducation soit au domaine du jardinage pour sa pertinence et pour identifier les améliorations à y apporter pour le rendre exploitable.

BIBLIOGRAPHIE

Liste des ouvrages cités

[Aamodt 93]

Aamodt A., "Explanation-Driven Case-Based Reasoning", *Topics in Case-Based Reasoning, First European Workshop*, EWCBR-93, Kaiserslautern, Germany, November 1993, p.274-288.

[Aamodt&Plaza]

Aamodt A., Plaza E., "Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approach", *Artificial Intelligence Communications*, 7(1), p.39-59.

[Agre 95]

Agre G., "An approach to Integration of Rule-Based and Case-Based Reasoning", *Problems of engineering cybernetics and Robotics*, n°42, Sofia, 1995, p.40-49.

[Aleven&Ashley 96]

Aleven V. and Ashley K.D., "How Different Is Different ? Arguing About the Significance of Similarities and Differences", *Advances in Case-Based Reasoning, Third European Workshop, EWCBR-96*, Lausanne, Switzerland, Proceedings, Smith I.& Faltings B. (Eds.), Lecture Notes in Artificial Intelligence 1168, November 1996, p.1-15.

[Ashley&Aleven 93]

Ashley K.D. et Aleven V., "A logical Representation for Relevance Criteria", *Topics in Case-Based Reasoning, First European Workshop, EWCBR-93*, Kaiserslautern, Germany, November 1993, p.338-352.

[Barber *et al.* 92]

Barber J., Bhatta S., Goel A., Jacobson M., Pearce M., Penberthy L., Shankar M., Simpson R. et Stroulia E., "AskJef: Integrating Case-Based and Multimedia

Technologies for Interface Design Advising", *Proceedings Second International Conference on Artificial Intelligence in Design*, Pittsburgh, June 1992, p.457-476

[Bartholomew 84]

Bartholomew M., *Méthode de jardinage en carrés*, Édition française Edicompo Inc., Ottawa, Horizon Rodale Press, 1984.

[Baumewerd-Ahlmann *et al.* 90]

Baumewerd-Ahlmann A., Jaschek P., Kalinski J., Lehmkuhl H., *Using KADS for Generating Explanations in Environmental Planning*, Forschungsberitch Nr.374, University of Dortmund, Federal Republic of Germany, 1990.

[Baumewerd-Ahlmann *et al.* 91a]

Baumewerd-Ahlmann A., Jaschek P., Kalinski J., Lehmkuhl H., "Embedding Explanations into Model-Based Knowledge Engineering - Improved decision Support in Environmental Impact Assessment-", *Proceedings of the First International Conference on Knowledge Modeling & Expertise Transfer*, Sophia-Antipolis, French Riviera, France, IOS Press, April 22-24, 1991, p.269-284.

[Baumewerd-Ahlmann *et al.* 91b]

Baumewerd-Ahlmann A., Jaschek P., Kalinski J., Lehmkuhl H., "Using KADS for Generating Explanations in Environmental Impact Assessment", *Proceedings of the Fourth International Conference on Human-Computer Interaction, Congress II: Design and Implementation of Interactive Systems*, vol.2, 1991, p.866-870.

[Bento *et al.* 94a]

Bento C., Exposto J., Francisco V. et Costa E., "Experimental Study of an Evaluation Function for Cases Imperfectly Explained", *Advances in Case-Based Reasoning. Proceedings Second European Workshop, EWCBR-94*, Chantilly, France, November 1994, p.33-44.

[Bento *et al.* 94b]

Bento C., Macedo L. et Costa E., "Reasoning with cases Imperfectly Described and Explained", *Advances in Case-Based Reasoning : Proceedings Second European Workshop, EWCBR-94*, Chantilly, France, November 1994, p.45-59.

[Bourcier *et al.* 94]

Bourcier F., Gréboval M.H., Kassel G. et Trigano P., "Construction d'une explication et génération en langue naturelle: une étude de cas", *RFIA'94, 9ème congrès*, Paris, 1994.

[Bourcier&Gréboval 92]

Bourcier F. et Gréboval M.H., "Représentation du contenu d'une explication", *Premières rencontres nationales des jeunes chercheurs en Intelligence Artificielle, IRISA,* Rennes, Septembre 1992, p.52-65.

[Buchanan&Shortliffe 84]

Buchanan B.G., Shortliffe E.H., *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project,* Addison-Wesley Publishing Company, 1984.

[Capus&Tourigny 98a]

Capus L., Tourigny N., "Road Safety Analysis: A Case-based Reasoning Approach", *Transportation Research Board, 77th Annual Meeting (TRB'98),* 11-15 Janvier 1998, Washington D.C, 1998.

[Capus&Tourigny 98b]

Capus L., Tourigny N., "Systèmes utilisant le raisonnement par cas : stratégie d'indexation", *INFOR,* vol.36 (1/2), Février/Mai 1998.

[Chandrasekaran *et al.* 89]

Chandrasekaran B., Tanner M.C., Josephson J.R., "Explaining Control Strategies in Problem Solving", *in IEEE Expert* , 1989, p.9-24.

[Cox&Ram 94]

Cox M.T. et Ram A., "Interacting Learning-Goals : Treating Learning as a Planning Task", *Advances in Case-Based Reasoning, Proceedings Second European Workshop, EWCBR-94,* Chantilly, France, November 1994, p.61-74.

[David 95]

David J.M., "Les systèmes experts de seconde génération ou de l'importance de la modélisation dans la construction de systèmes à base de connaissances", *Technique et science informatiques,* Volume 14, n°4, 1995, p.435-471.

[David *et al.* 93]

David J.M., Krivine J.P., Simmons R., "Second generation Expert Systems: A Step Forward in Knowledge Engineering", *David J.M., Krivine J.P., Simmons R. (eds); Second Generation Expert Systems,* Springer-Verlag, p.3-45, 1993.

[De Hoog *et al.* 94]

De Hoog R., Benus b., Metselaar C., Vogler M., Menezes W., *Organisation Model: model Definition Document, document KADS-II/M6/UvA/041/3.0*, University of Amsterdam, June 1994.

[Dhaliwal 98]

Dhaliwal J.S., "Design and Use of Explanation Facilities", *The handbook of Applied Expert systems*, Edited by Jay Liebowitz, CRC Press LLC, 1998.

[Dieng 93]

Dieng R., "Méthodes et outils d'acquisition des connaissances", *L'ergonomie dans la conception des projets informatiques*, Jean-Claude Sperandio, Octares editions, 1993, p.335-412.

[Duursma *et al.* 94]

Duursma C., Olle O., Ulf S., *Task Model Definition and Task Analysis Process, KADS-II/M5/VUB/RR/004/2.0*, Vrije univeresiteit Brusseil, AI Laboratory, August 1994.

[Gagnon 84]

Gagnon Y., *Introduction au Jardinage Écologique*, St-Didace, 1984.

[Gebhardt *et al.* 97]

Gebhardt F., Voß A., Gräther W., Schmidt-Belz B., *Reasoning with complex cases*, Kluwer Academic Publishers, Germany, 1997.

[Goel *et al.* 97]

Goel A.K., Gomez de Silva Garza A., Grue N., Murdock J.W. et Recker M.M., "Functional Explanations in Design", *IJCAI'97, Workshop on Modelling and Reasoning about Function.*, Intelligence and Design Group, College of Computing, Georgia Institute of Technology, 1997.

[Goel *et al.* 96a]

Goel A.K., Gomez de Silva Garza A., Grue N., Murdock J.W., Recker M.M. et Govindaraj T., "Explanatory Interface in Interactive Design Environments", *Fourth International Conference on AI in Design*, Stanford University, June 1996.

[Goel *et al.* 96b]

Goel A. et Murdock J.W., "Meta-Cases : Explaining Case-Based Reasoning", *Advances in Case-Based Reasoning. Proceedings Third European Workshop, EWCBR-96*, Lausanne, Switzerland, November 1996, p.150-163.

[Gonzalez&Dankel 93]

Gonzalez A.J., Dankel D.D., *Engineering of Knowledge-Based Systems,* Prentice Hall, Englewood Cliffs (New Jersey), 1993.

[Gréboval 92]

Gréboval C., "SATIN: Un modèle conceptuel opérationnel", *Premières rencontres nationales des jeunes chercheurs en Intelligence Artificielle, IRISA,* Rennes, Septembre 1992, p.132-142.

[Greer *et al.* 94]

Greer J.E., Falk S., Greer K.J. et Bentham M.J., "Explaining and justifying recommendations in an agriculture decision support system", *Computers and electronics in agriculture,* vol. 11 (2-3), 1994, p.195-214.

[Grimnes&Aamodt 96]

Grimnes M. et Aamodt A., "A two layer case-based reasoning architecture for medical image understanding", *Advances in Case-Based Reasoning, Third European Workshop, EWCBR-96,* Lausanne, Switzerland, Proceedings, Smith I.& Faltings B. (Eds.), Lecture Notes in Artificial Intelligence 1168, November 1996, p.164-178.

[Kamp 93]

Kamp G., "Integrating Semantic Structure and Technical Documentation in Case-Based Service Support Systems", *Topics in Case-Based Reasoning, First European Workshop, EWCBR-93,* Kaiserslautern, Germany, November 1993, p.392-403.

[Kassel 93]

Kassel G., "Réflexion au niveau connaissance dans AIDE", *Rapport Interne HEUDIASYC Nde. 93/01/DI,* Université de Compiègne, janvier 1993.

[Kolodner 93]

Kolodner J., *Case-Based Reasoning,* Morgan Kaufman Publishers, San Mateo, CA, États-Unis, 1993.

[Leake 95]

Leake D., *Abduction, Experience, and Goals: A Model of Everyday Abductive Explanation,* Computer Science Department, Indiana University, Février 1995.

[López de Mántaras&Plaza 97]

López de Mántaras R., Plaza E., "Case-Based reasoning : An Overview", *AI Communication Journal,* Vol. 10, n°1, 1997, p.21-29.

[Luger&Stubblefield 98]

Luger G.F. et Stubblefield W.A., *Artificial Intelligence : Structures and Strategies for Complex Problem Solving*, Third Edition, Addison Wesley Longman, Inc, 1998.

[Macedo *et al.* 96]

Macedo L., Pereira F.C., Grilo C. et Cardoso A., "Plans as Structured Networks of Hierarchically and Temporally Related Case Pieces", *Advances in Case-Based Reasoning, Proceedings, Third European Workshop, EWCBR-96*, Lausanne, Switzerland, November 1996, p.234-248.

[Madhavan 94]

Madhavan R.K., "Goal-Based Reasoning for Securities Analysis", *AI Expert*, Vol. 9, Iss 2, USA, February 1994, p.22-29.

[Martin 94]

Martin P., *La méthodologie d'acquisition des connaissances KADS et les explications*, Rapport de Recherche, INRIA, Sophia Antipolis, 1994.

[Möller 95]

Möller J.U., "Operationalisation of KADS models by using conceptual graph modules", *9th Knowledge Acquisition for Knowledge Based Systems Workshop*, 1995, Banff, Calgary, 1995.

[Moore 89]

Moore J.D., *A reactive approach to explanation in expert and advice giving systems*, Thèse de doctorat, University of California, 1989.

[Moore&Swartout 88]

Moore J.D., Swartout W.R., *Explanation in Expert Systems : A Survey ,ISI Research Report, ISI/RR-88-228*, University of Southern California, December 1988.

[Muñoz-Avila&Hüllen 96]

Muñoz-Avila H. et Hüllen J., "Feature Weighting by Explaining Case-Based Planning Episodes", *Advances in Case-Based Reasoning. Proceedings Third European Workshop, EWCBR-96*, Lausanne, Switzerland, November 1996, p.280-94.

[Newell 82]

Newell A., "The Knowledge Level", *Artificial Intelligence*, 1982.

[Paris *et al.* 88]

Paris C.L., Wick M.R., Thompson E.B., "The line of Reasoning versus the line of Explanation", *Proceedings of the AAAI'88 Workshop on Explanation*, 1988, p.4-7.

[Quinlan 86]

Quinlan J.R., "Induction of decision trees", *Machine Learning*, volume 1, 1986, p.81-106.

[Quinlan 93]

Quinlan J.R., *C4.5 Programs for Machine Learning*, Morgan Kaufmann Publishers, San Mateo (Californie), 1993.

[Reategui *et al.* 95]

Reatigui E., Campbell J.A. et Borghetti S., "Using a Neural Network to Learn General Knowledge in a Case-Based System", *Case-Based Reasoning Research and Development. Proceedings in First International Conference, ICCBR-95*, Sesimbra, Portugal, October 1995, p.528-537.

[Reategui *et al.* 97]

Reategui E.B., Campbell J.A. et Leao B.F., "Combining a neural network with case-based reasoning in a diagnostic system", *in Artificial Intelligence in Medicine*, vol 9, N 1, 1997, p.5-27.

[Riesbeck&Schank 89]

Riesbeck C.K., Schank R.C., *Inside case-based reasoning*, Lawrence Erlbaum Associates, Publishers, Hillsdale, New Jersey, 1989.

[Robert 94]

Robert P., *Le nouveau Petit Robert, dictionnaire alphabétique et analogique de la langue française*, Dictionnaires Le Robert, Paris, 1994.

[Schank 82]

Shank R., *Dynamic memory : A theory of learning in computers and people*,: Cambridge University Press, New York, 1982.

[Schank *et al.* 94]

Schank R.C., Kass A., Riesbeck C., *Inside Case-Based Explanation*, The Institute for the Learning Sciences, North-western University, Lawrence Erlbaum Associates, Publishers, Hillsdale, New Jersey, Hove, UK, 1994.

[Schreiber *et al.* 93]

Schreiber G., Wielinga B., Breuker J., *KADS : A Principled Approach to Knowledge-Based System development*, volume II, Academic Press, Boston, 1993.

[Schreiber *et al.* 94]

Schreiber G., Wielinga B., De Hoog R., Akkermans H., Van de Velde W., "CommonKADS: A Comprehensive Methodology for KBS Development", *IEEE Expert*, vol. 9, n°6, Décembre 1994, p.28-37.

[Scott&Weckert 97]

Scott J.& Weckert J., "Helping the User to Understand: Dynamic Explanations", *AI Applications, vol. 11, No 3*, 1997, p.19- 29.

[Simian&Tourigny 96]

Simian G., Tourigny N., *IFT-63677 Conception et architecture des systèmes experts, Support de cours*, Département d'Informatique, Université Laval, Canada, Hiver 1996.

[Sprenger 93]

Sprenger M., "Explanation strategies for KADS-based expert systems", *New Concepts in Natural Language Generation Planning, Realisation and Systems*, Edited by Helmut H. and Zock M., Communication in Artificial Intelligence series, 1993, p.27-56.

[Swartout&Moore 93]

Swartout W.R., Moore J.D., "Explanation in Second Generation Expert Systems", dans [David *et al.* 93], 1993.

[Tansley&Hayball 93]

Tansley D.S.W., Hayball C.C., *Knowledge-Based Systems Analysis and Design, A KADS Developer's Handbook*, Software&Systems Engineering, Advanced Technology centre, BNR Europe Limited, Prentice Hall International, Great Britain, 1993.

[Tourigny 98]

Tourigny N., "Systèmes d'aide à l'étude de la sécurité routière : Vers des outils hybrides, ouverts et intelligents", *Recherche Transports Sécurité*, n°59, Avril-Juin 1998, p.58-79.

[Van de Velde 94]

Van de Velde W., *Design Model and Process, document KADS-II/M7/VUB/RR/064/2.1*, Vrije Universiteit Brussel, October 1994.

[Verrière 97]

Verrière H., *Les explications dans les systèmes à base de connaissances utilisant le raisonnement par cas*, Rapport de DEA en Représentation de la Connaissance et Formalisation du Raisonnement, Intelligence Artificielle, Université Paul Sabatier, Toulouse, France, juin 1997.

[Verrière *et al.* 97]

Verrière H., Tourigny N., Simian G., "Modèles d'explcations dans les systèmes à base de connaissances utilisant le raisonnement par cas", *65ème congrès de l'ACFAS*, Trois-Rivières, Canada, 12-16 Mai 1997.

[Verrière&Tourigny 98]

Verrière H., Tourigny N., "Modélisation d'explications : une application à l'analyse de la sécurité routière", *66ème congrès de l'ACFAS*, Québec, Canada, 11–15 Mai 1998.

[Wærn *et al.* 93]

Wærn A., Höök K., Gustavsson R., Holm P., *The CommonKADS Communication Model, document KADS-II/M3/SICS/TR/006/V.2.0*, Swedish Institute of Computer Science, December 1993.

[Wærn&Gala 93]

Wærn A., Gala S., *The CommonKADS Agent Model, document KADS-II/M4/TR/SICS/005/V2.0*, Swedish Institute of Computer Science, December 1993.

[Watson 97]

Watson I., *Applying Case-Based Reasoning, techniques for Enterprise Systems*, Morgan Kaufmann Publishers, Inc, San Francisco, California, 1997.

[Weber 94]

Weber G., "Examples and Remindings in a Case-based Help System", *Advances in Case-Based Reasoning. Proceedings Second European Workshop, EWCBR-94*, Chantilly, France, November 1994, p.165-177.

[Wick&Thomson 92]

Wick M. R., Thomson W.B., "Reconstructive Expert System Explanation", *Artificial Intelligence*, n°54 (1-2), p.33-70, Mars 1992.

[Wielinga *et al.* 92]

Wielinga B.J., Schreiber G., Breuker J., "KADS : a modelling approach to knowledge engineering ", *Knowledge Acquisition*, vol. 4, p.136-145, 1992.

[Wielinga 94]

Wielinga B.J., *Expertise Model Definition Document*, Document KADS-II/M2/UvA/026/5.0, June 1994.

Ouvrages non cités

[Leake 96]

Leake D.B., *Case-Based Reasoning, Experiences, Lessons&Future Directions*, AAAI Press/The MIT Press, Massachusetts Institute of Technology, Massachusetts, 1996.

[McGuinness 96]

McGuinness D.L., *Explaining Reasoning in Description Logics*, Thesis Memory, New Brunswick Rutgers, The State University of New Jersey, 1996.

[Tourigny 94]

Tourigny N., *La génération automatique de descriptions de systèmes dynamiques*, Thèse de doctorat, département d'informatique et de recherche opérationnelle, Université de Montréal, Novembre 1994.