République Algérienne Démocratique et Populaire.
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique.

Université Saad Dahlab, Blida
USDB.

Faculté des sciences.
Département informatique .

Mémoire pour l'obtention
d'un diplôme d'ingénieur d'état en informatique.
Option : Intelligence Artificielle

Sujet :

Modélisation de l'explication dans les systèmes à base de connaissances : application sur un moteur d'inférences d'ordre 0+

Présenté par : Mr. CHIKHI Nacim Fateh
Mr. ABBAD Med Amine

Promoteur : Dr. OUKID S.
Encadreur : Dr. SAIDI M.

Organisme d'accueil : E.P.I.T.A (Alger)

Soutenue le: 02 juillet 2005 , devant le jury composé de :

Nom. président du jury, grade, organisme	**Président**
Nom examinateur 1, grade, organisme	**Examinateur**
Nom examinateur 2, grade, organisme	**Examinateur**

- Numéro/Juin 2005 -

République Algérienne Démocratique et Populaire.
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique.

Université Saad Dahlab, Blida
USDB.

Faculté des sciences.
Département informatique .

Mémoire pour l'obtention
d'un diplôme d'ingénieur d'état en informatique.
Option : Intelligence Artificielle

Sujet :

Modélisation de l'explication dans les systèmes à base de connaissances : application sur un moteur d'inférences d'ordre 0+

Présenté par : Mr. CHIKHI Nacim Fateh
Mr. ABBAD Med Amine

Promoteur : Dr. OUKID S.
Encadreur : Dr. SAIDI M.

Organisme d'accueil : E.P.I.T.A (Alger)

Soutenue le: 02 juillet 2005 , devant le jury composé de :

Nom. président du jury, grade, organisme	**Président**
Nom examinateur 1, grade, organisme	**Examinateur**
Nom examinateur 2, grade, organisme	**Examinateur**

- Numéro/Juin 2005 -

SOMMAIRE

INTRODUCTION GENERALE

CHAPITRE I Concepts généraux sur les systèmes à bas de connaissances

CHAPITRE II Explication dans les systèmes à base de connaissances

CHAPITRE III Conception et réalisation du système SDPTV

Remerciements

Avant tout, nous remercions Allah le Tout-Puissant qui nous a aidé à réaliser ce projet.

Nous tenons à remercier vivement, notre promoteur, Mr SAIDI Mustapha pour son aide très fructueuse, ses précieux conseils, sa patience, sa disponibilité et son dévouement total.

Nous remercions Mme OUKID-KHOUAS S., co-promotrice, pour ses précieux conseils, et qui a su nous faire profiter de sa grande expérience.

Nous remercions Mme BENSETITI S., chef du département Informatique à l'Université de BLIDA, qui a su créer une bonne ambiance de travail au sein du département.

Nous remercions Mr CHIKHI M.L. (mon père), pour sa précieuse collaboration à la mise en œuvre de la base de règles du système SDPTV.

Nous remercions tout le personnel de E.P.I.T.A pour leur sympathie et bienveillance durant notre stage.

Dédicace

Je dédie ce travail :

A mes très chers parents qui m'ont soutenu durant toute ma carrière et qui m'ont aidé à surmonter les moments difficiles,

A mes sœurs Amina et Imane,

A toute ma famille,

A mon binôme Amine et à toute sa famille,

A tous mes amis,

A tous les enseignants et étudiants du département Informatique de l'Université de BLIDA.

CHIKHI Nacim Fateh

Dédicace

Je dédie ce mémoire :

A mes parents qui ont tant attendu ce moment.
A mon frère et mes sœurs.
A toute ma famille.
A mon ami Nacim et sa famille.
A tous mes amis.

Amine.

RESUME

Si ces dernières années, l'intérêt des explications dans les systèmes à base de connaissances était jugé secondaire, il apparaît clairement maintenant que le succès des systèmes à base de connaissances repose sur la qualité des explications que ceux-ci peuvent fournir. La plupart des recherches sur ce sujet se sont tout d'abord concentrées sur des « explications basées sur la trace » pour expliquer le processus de raisonnement. Les efforts ont principalement porté sur la transparence du comportement du résolveur et sur les stratégies de résolution. Notre travail dans ce rapport, consiste d'une part à étudier de façon très détaillée l'ensemble des travaux dans ce domaine et d'autre part de développer un système expert explicatif basé sur la trace de résolution. Afin d'étayer nos idées sur un cas réel, nous avons établi avec l'aide de l'ingénieur de la connaissance, une base de connaissances. Plusieurs types d'explications ont été proposées : explication en cours de session et en fin de session, explication positive et explication négative. Afin de tenir compte du niveau de l'interlocuteur, nous avons adjoint un module qui prend en considération le niveau de l'interlocuteur lors de la production d'explication.

Mots clés :
Intelligence artificielle, Systèmes à base de connaissances, Systèmes experts, explication, raisonnement, représentation des connaissances, moteur d'inférences d'ordre 0+, SDPTV, diagnostic de pannes de télévisions, trace de raisonnement.

INTRODUCTION GENERALE

Si d'une façon générale, l'intérêt manifesté pour la production d'explication dans les systèmes à base de connaissances est clair, la nature des explications à produire, les méthodes de développement des systèmes explicatifs, la modélisation de l'utilisateur dans l'explication, sont autant de questions qui ont suscité de nombreux travaux dans des directions très diverses [Kassel, Safar, Saîdi, Lemaire, Clancey, ...].

Une première catégorie de travaux [Swartout et Chandrasekaran] s'intéresse au manque de connaissances nécessaires pour la production d'explication. L'idée de base de ces travaux est que plus les connaissances à expliquer sont explicitées et représentées clairement, plus la tâche d'explication en est facilitée.

Une deuxième catégorie de travaux [Jiménez, Saîdi, Cawsey] considèrent l'explication comme une tâche de résolution de problèmes à part entière, mettant en œuvre son propre raisonnement. Un système explicatif doit savoir interpréter les questions posées par l'utilisateur, accepter les interruptions de ce dernier pendant la production d'explication, pouvoir revenir sur une explication déjà produite.

Le thème central de notre projet est l'explication dans le cadre des moteurs d'inférences d'ordre 0+. Notre travail consiste d'une part à comprendre la problématique de l'explication dans les systèmes à base de connaissances et d'autre part de concevoir et de réaliser le système SDPTV (Système de Diagnostic de Pannes de TéléVision) qui construit et transmet une explication à l'utilisateur final d'un système à base de connaissances. Cette explication est produite par un module d'explication, qui travaille à partir d'une trace de raisonnement.

Ce travail est divisé en quatre parties :

1. La première traite des concepts généraux sur les systèmes à base de connaissances. Après la définition de « l'intelligence artificielle » et la présentation des différents domaines d'application, nous abordons les systèmes à base de connaissances.

2. La seconde partie est consacrée à l'explication. Après une étude bien détaillée sur l'état actuel des recherches en explication, nous citons les principaux axes de recherche.

3. La troisième partie décrit le système SDPTV. Cette partie est axée sur deux axes principaux : Le système expert de résolution et le système d'explication.

4. Une maquette a été mise en œuvre, qui implante ce qui a été décrit dans le chapitre III. Un exemple de construction d'explication achève cette partie.

Chapitre I

Concepts généraux sur les systèmes à base de connaissances

1. Introduction

L'intelligence artificielle (IA) a vu son importance s'accroître considérablement par sa capacité à s'attaquer à de nouvelles classes de problèmes, différents de ceux traités par l'informatique classique. Ces problèmes relèvent d'activités humaines variées (perception, prise de décision, planification, diagnostic, interprétation de données, compréhension du langage, conception) et présentent la particularité commune de nécessiter une exploitation raisonnée d'une grande quantité de connaissances, pour l'essentiel spécifiques du domaine étudié et acquises auprès d'experts.

La conception de systèmes à bases de connaissances (SBC) capables de réaliser des fonctions de raisonnement symbolique constitue ainsi à l'heure actuelle une part importante des recherches et des développements en IA. De tels systèmes nécessitent en particulier une représentation adéquate des connaissances mises en jeu ainsi que des mécanismes efficaces d'exploitation de ces connaissances ou de raisonnement.

Les SBC recouvrent un vaste domaine. Ce chapitre en introduit les généralités, les principes de fonctionnement et quelques exemples d'applications. Nous présentons ainsi une brève synthèse sur l'IA avant de décrire l'architecture générale d'un système expert. Par la suite seront abordés les différents formalismes de représentation des connaissances.
Enfin, nous indiquons des exemples d'application des SBC, puis nous exposons les difficultés rencontrées par les premiers systèmes experts dits de première génération (SE1G).

2. Qu'est ce que l'intelligence artificielle

2.1. Définitions

Il existe plusieurs définitions de l'intelligence artificielle (IA), ces dernières peuvent être regroupées en quatre catégories (voir figure I.1). Ces définitions varient selon deux dimensions principales. Les définitions du dessus se focalisent davantage sur l'aspect **processus de pensée**, alors que celles du dessous se focalisent sur **l'aspect opérationnel** et le **comportement du système** uniquement. Par ailleurs, les définitions de gauche mesurent le succès de l'entreprise en termes de **performances similaires à l'humain**, alors que celles de droite se comparent à un **concept idéal** d'intelligence, que nous appellerons «rationnalité».

Au niveau opérationnel, et ingénieriste, c'est la dernière définition (4) – agir rationnellement – qui est souvent retenue. En effet, au niveau opérationnel, on se soucie fort peu du fait que la machine pense réellement, ou qu'elle soit similaire à l'homme. Cette approche est souvent appelée «agent rationnel».

La définition (3) – agir humainement – s'apparente au test de Turing*, en se mesurant aux performances de l'homme.

---

* Le test de Turing s'appuie sur un jeu développé par Alain Turing dans l'année 1950. Le principe de base est qu'une personne est connectée par le biais d'un terminal à un homme et à une machine : en posant des questions, la personne tente de distinguer si c'est l'homme ou la machine qui répond. Si au bout de cinq minutes de conversation, il ne parvient pas à faire la distinction, l'ordinateur est considéré comme vainqueur.

(1) Des systèmes qui pensent comme l'homme	**(2) Des systèmes qui pensent rationnellement**
«L'effort excitant visant à bâtir des machines qui pensent ... des machines avec un esprit, dans le plein sens du terme» (Haugeland, 1985) «L'automatisation des activités qui sont associées à la pensée (human thinking), c'est-à-dire des activités telles que la prise de décision, la résolution de problèmes, l'apprentissage...» (Bellman, 1978)	«L'étude des facultés mentales en utilisant des modèles computationnels» (Charniak & McDermott, 1985) «L'étude des processus computationnels qui rendent possible la perception, la raison et l'action» (Winston, 1992)
(3) Des systèmes qui agissent comme l'homme	**(4) Des systèmes qui agissent rationnellement**
«L'art de créer des machines qui réalisent des actions exigeant de l'intelligence de la part de l'homme» (Kurzeil, 1990) «L'étude des moyens computationnels qui pourraient amener les ordinateurs à réaliser des tâches pour lesquelles les humains sont plus performants» (Rich & Knight, 1991)	«Le domaine d'étude qui essaie d'expliquer et émuler un comportement intelligent en terme de processus computationnels» (Schalkoff, 1990) «Le domaine des sciences computationnelles qui s'intéresse à l'automatisation de comportements intelligents» (Luger & Stubblefield, 1993)

Figure I.1 - Quelques définitions de l'intelligence artificielle regroupées en quatre catégories.

La définition (1) – penser humainement – est davantage du ressort des sciences cognitives, qui élaborent des modèles perceptifs, cognitifs, ou du comportement humain, et observent si ce modèle colle à la réalité expérimentale.

La définition (2) s'apparente à l'élaboration de systèmes idéaux, possédant une intelligence rationnelle. Cette voie est suivie par la tradition «logique» de l'intelligence artificielle, qui espère obtenir des systèmes intelligents basés sur la logique formelle. Bien entendu, les systèmes (2), qui représentent des concepts idéaux de l'intelligence et, de ce fait, ne se comparent pas à l'homme – et refusent le test de Turing – se heurtent à des problèmes importants d'interprétation et de validation.

2.2. Historique

La naissance officielle de l'intelligence artificielle remonte au mois d'août 1956,à la conférence tenue au Dartmouth College (Hanover, New Hampshire) au cours de laquelle J. McCarthy donna à la discipline le nom d'Intelligence Artificielle, et se proposa d'explorer en compagnie de M. Minsky, A. Newell, H. Simon, C. Shanon, et d'autres, la conjecture selon laquelle tout ce qui relève de l'intelligence peut être décrit avec suffisamment de précision pour être réalisé par une machine.

Les premiers programmes d'intelligence artificielle datent également de cette période : un démonstrateur de théorèmes en logique des propositions (« The Logic Theorist de Newell, Shaw, Simon en 56), un programme jouant aux échecs (des mêmes auteurs en 57), un interprète du langage Lisp (McCarthy 58).

Les premières années furent marquées par quelques apports et surtout beaucoup d'optimisme : comprendre un texte en langue naturelle, battre un champion aux échecs semblaient des tâches à portée immédiate. L'estimation était fausse, il fallut au contraire une dizaine d'années de maigres résultats pour bien appréhender les difficultés énormes de ces problèmes et élaborer des stratégies d'approche.

La fin des années soixante a été plus productive et a amené un véritable décollage de la discipline. On y trouve entre autres résultats le « principe de résolution », l'algorithme A* et des programmes tels que GPS.

La décennie suivante fut riche en contributions. Elle amena les bases essentielles en représentation de la connaissance et les premiers résultats sur les problèmes difficiles tels que la compréhension des langues naturelles. A cette époque apparurent également des méthodes et des outils pour le développement d'applications réelles : systèmes experts, perception et vision par ordinateur, robotique, compréhension de la parole et des langues naturelles en contextes restreints, ...etc.

Les années 80 concrétisèrent ces résultats. Au niveau de la recherche en intelligence artificielle, on dispose d'une plus grande maîtrise des outils et des résultats fondamentaux acquis, et d'une perception plus précise des problèmes scientifiques ouverts et de leur difficulté. Au niveau des applications on assiste aux premières commercialisations et à la maîtrise technologique correspondante.

Les tendances des années 90 sont difficiles à décrire car l'intelligence artificielle comprend de plus en plus de domaines. Mais nous pouvons dégager trois grandes tendances : "l'Intelligence Artificielle théorique" s'attache à décrire par des modèles mathématiques le fonctionnement du raisonnement et l'acquisition des connaissances, "l'Intelligence Artificielle expérimentale" teste des idées nouvelles (concernant la mémoire, l'apprentissage) au moyen de systèmes informatiques. "L'Intelligence Artificielle appliquée" crée des systèmes pour les besoins de l'industrie ou des services. Ce qui caractérise l'Intelligence Artificielle aujourd'hui est son interaction avec un grand nombre de disciplines scientifiques qui se présentent en deux groupes : les sciences "cognitives" avec les neurosciences, la psychologie, la linguistique, la sociologie, et les sciences "formelles" avec les mathématiques, la logique, la physique et l'informatique.

2.3. Domaines d'application

Les fondateurs de l'intelligence artificielle revendiquent pour leur discipline de nombreuses retombées indirectes, principalement en informatique. Cela va de l'exploitation des ordinateurs en temps partagé (apparue dans le AI Laboratory du MIT sept à huit ans avant la commercialisation de tels systèmes d'exploitation), jusqu'aux jeux vidéo et leurs techniques graphiques, en passant par les éditeurs de texte.

Les applications de l'intelligence artificielle relevant de son domaine propre sont aujourd'hui nombreuses : les systèmes experts, les interfaces homme-machine, la vision par

ordinateur, la robotique, les diverses tâches assistées par ordinateur (conception, enseignement, ...etc.), la reconnaissance de la parole, le traitement du langage naturel, etc...

2.3.1 Programmation par contraintes

En programmation par contraintes ou CSP (Constraint Satisfaction Problem), contrairement aux systèmes experts, il ne s'agit pas d'appliquer des règles du type : « si ... alors ... », mais de gérer une combinatoire, c'est-à-dire, parmi un très grand nombre de cas, de trouver en un temps limité ceux qui répondent aux impératifs du problème.
En milieu industriel, on peut rencontrer des problèmes qui s'énoncent, par exemple, sous les formes suivantes : « La tâche A doit être effectuée avant la tâche B » ; « X hommes doivent être affectés à N postes » ; « La durée des processus A et B ne doit pas dépasser 2 heures » ...

Depuis le début des années 90, de nombreux outils de programmation par contraintes sont apparus sur le marché, dont les principaux sont : *Prolog III*, de la société PrologIA ; *Charme* de Bull ; *Decision Power* d'ICL ; etc...

2.3.2 Raisonnement basé sur le cas

Le raisonnement basé sur le cas se présente comme une alternative aux systèmes experts lorsque le domaine considéré est faiblement théorisé ?

Souvent, lorsqu'il s'agit d'établir un diagnostic industriel ou médical, ou lorsque l'on veut identifier un phénomène ou un objet (reconnaissance de formes, de risques, etc.), on est amené à se poser la question suivante : « Ai-je déjà vu un problème semblable et, si oui, qu'est-ce que j'ai fait pour le résoudre ? ». Un nouveau problème est résolu en identifiant sa similitude avec un problème passé et en adaptant sa solution pour résoudre le nouveau problème. Pour cela, il faut disposer d'une base de cas, pouvoir y accéder, l'élargir et la mettre à jour facilement. La solution est fournie par une technique d'intelligence artificielle : le *raisonnement basé sur le cas* ou *CBR (Case Based Reasoning)*.

ReMind, de Cognitive Systems, est utilisé, par exemple, pour retrouver les incidents dans les centrales nucléaires. *CBR Express*, d'Inference, est appliqué notamment à la maintenance téléphonique de matériel et logiciel, ainsi qu'à l'aide à la vente ...

2.3.3 Reconnaissance de la parole

Le traitement de la parole représente certainement l'un des éléments clés des systèmes informatiques de demain. Eminemment utile dans la relation homme-machine, la reconnaissance vocale dépasse largement le cadre de l'informatique par ses applications. Elle peut équiper des automobiles qui réagiront à un certain nombre d'ordres, elle assiste les handicapés moteurs, il existe déjà des machines-outils à commande numérique répondant à la voix, ainsi que des téléphones capables de composer eux-mêmes le numéro demandé oralement, directement dans le combiné.

La reconnaissance de la parole se situe à l'intersection de nombreux domaines tels que l'acoustique, l'électronique, la phonétique, la sémiologie, etc.

2.3.4 Traitement du langage naturel

La programmation des ordinateurs en vue de la compréhension des langues naturelles (on parle de « traitement du langage naturel ») a pour objectif de rendre les machines plus conviviales, de faciliter la communication entre l'homme et la machine. Elle englobe la compréhension de textes écrits et oraux.

Le traitement du langage naturel a de nombreuses applications, potentielles ou opérationnelles :

- La reconnaissance de la parole, où le traitement du langage naturel apporte des éléments permettant d'envisager la reconnaissance de mots enchaînés ;
- L'indexation automatique de textes ou la recherche d'informations dans des textes non structurés, écrits ou oraux, ce qui implique la reconnaissance de termes ou d'idées.
- La production de textes en langage naturel, avec les applications particulières de la création de récits, de l'élaboration de résumés, de la rédaction de lettres, etc.
- Le contrôle des systèmes informatiques ou automatiques, notamment l'accès à des bases de données, le dialogue avec un système expert ou avec un robot.
- Enfin, la traduction automatique, qui inclut la compréhension de texte dans la langue source et la génération de texte dans la langue cible.

2.3.5 Vision par ordinateur

La vision par ordinateur permet, à partir de l'image d'un objet ou d'une scène réelle, d'en déduire des données exploitables par une machine.

Combinant optique, électronique, traitement du signal et informatique, la vision par ordinateur substitue au système visuel humain un ensemble informatisé (matériel + logiciel) capable d'intégrer l'acquisition d'images, leur traitement et la prise de décision adaptée au cas identifié.

La vision par ordinateur a pour objectif, non de se substituer à l'homme, mais de rendre les machines indépendantes de celui-ci, notamment dans les environnements sensibles, où la présence de l'homme serait nuisible (salles blanches, dans l'industrie électronique par exemple), ou dangereux (milieux radioactifs, par exemple).

2.3.6 Traduction automatique

Le problème de la traduction automatique se pose dans les termes suivants : étant donné, en entrée, un texte rédigé dans la « langue source », le programme doit fournir, comme résultat, un texte élaboré dans la « langue cible ».

Le programme *Systran* mis au point au cours des années soixante par Peter Toma, à l'université de Georgetown est le premier système opérationnel et commercial de traduction automatique. Conçu d'abord pour la traduction russe-anglais, ce système a ensuite été étendu à la plupart des langues européennes.

2.3.7 Robots et systèmes autonomes

Les robots ont quitté le domaine de la science fiction pour venir peupler les ateliers, les usines, les unités de production, etc... Les premières applications de la robotique datent des années soixante. Les robots ont commencé par investir l'industrie automobile puis celles de l'électronique et de l'électroménager.

Dans les premières années, il ne s'agissait généralement que d'automates mécaniques programmés pour accomplir une tâche donnée : soudage, découpe, peinture, etc... Peu à peu, les robots ont été dotés d'intelligence : des capteurs leurs permettent, à l'instar des yeux humains, de se déplacer et d'évoluer dans un milieu mal connu ou inconnu.

2.3.8 Aider intelligemment

L'IA sert non seulement à simuler ou reprendre le raisonnement, mais aussi à rassembler, conserver et rendre accessible les connaissances disponibles dans un domaine donné. Ces connaissances peuvent se présenter comme une « aide en ligne », accessible sur un poste de travail, en l'occurrence un ordinateur. C'est ainsi que toutes les tâches informatiques peuvent bénéficier de l'IA, en particulier celles que l'on désigne sous le sigle de XAO ou « X (n'importe quoi) assisté par ordinateur », qui devient dés lors XIAO ou « X intelligemment assisté par ordinateur ».

De telles connaissances peuvent être utilisées dans un programme d'enseignement assisté par ordinateur, lequel est capable de s'adapter au niveau et aux besoins de l'élève, en lui fournissant les connaissances au fur et à mesure de ses demandes, et non d'une manière prévue par le programmeur une fois pour toutes. Les réalisations dans ce domaine sont nombreuses : tuteurs intelligents, EIAH (Environnements Informatiques pour l'Apprentissage Humain), micro-mondes, etc...

Nous remarquons donc qu'à l'heure actuelle, les domaines d'application de l'IA sont nombreux et divers, cela traduit indirectement un grand succès de cette discipline. Toutefois, elle doit en grande partie ce succès aux *systèmes experts*, qui, dés leurs apparition au début des années 70, ont eu un important impact sur celle-ci. En effet, ils furent la première vraie consécration de l'IA après plus de quinze ans de recherches.

3. Généralités sur les systèmes à base de connaissances

3.1. Introduction aux systèmes à base de connaissances

La conception de systèmes à bases de connaissances constitue un domaine majeur en intelligence artificielle (IA). De tels systèmes sont conçus pour atteindre les performances d'experts humains dans des domaines limités en exploitant un ensemble de connaissances acquises pour l'essentiel auprès de ces experts. Apparus au début des années 70, ils ont eu un impact certain sur l'intelligence artificielle ainsi qu'un retentissement médiatique parfois exagéré.

Le terme de système expert (SE) disparaît au profit du concept plus général de système à base de connaissances (SBC) que l'on retrouve dans divers champs d'activités de

l'intelligence artificielle (notons que tout au long de nos travaux, les appellations : systèmes experts et systèmes à base de connaissances sont employées comme synonymes). Ce concept est fondé sur *une séparation entre les connaissances nécessaires pour résoudre un problème et les mécanismes de raisonnement exploitant ces connaissances* (appelés selon les cas structures de contrôle, interpréteurs, moteurs d'inférence). Cette définition, illustrée par la figure I.2, montre la dualité connaissances/raisonnement qui vient compléter la structure algorithmique traditionnelle de l'informatique procédurale.

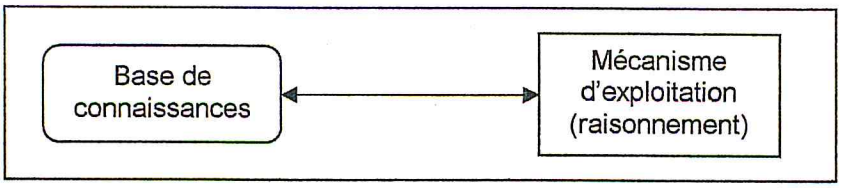


Figure I.2 - Schéma simplifié d'un système à base de connaissances

Cette indépendance entre connaissances et mécanisme de raisonnement permet une représentation des connaissances sous forme purement *déclarative*, c'est-à-dire sans lien avec la manière dont ces connaissances sont utilisées. Les avantages d'une telle représentation sont nombreux : possibilité de faire évoluer les connaissances du système sans avoir à agir sur le mécanisme de raisonnement, possibilité d'expliquer le raisonnement suivi par le système, etc..

Les systèmes à base de connaissances suscitent un important engouement dans divers milieux professionnels parce qu'ils permettent d'aborder l'informatisation de certaines fonctions intellectuelles qualifiées, telles que l'identification ou le diagnostic de situations, la prévision d'événements, la conception d'objets, la planification d'actions, qui sont très concrètement utiles dans l'activité des entreprises.

3.2. Architecture générale d'un système expert

Les systèmes experts sont en général composés d'une architecture à quatre modules (voir figure I.3) :

- La *base de connaissances.*
- La *base de faits*
- Le *moteur d'inférences.*
- L'*interface.*

3.2.1. Base de connaissances

La base de connaissances contient toutes les informations dont l'expert humain a besoin pour s'acquitter de son travail. Cela correspond à des faits, des règles.

Afin de favoriser une plus grande maniabilité de la connaissance, il est fréquent de la stocker déclarativement dans un langage clair, proche du langage naturel. Grâce à ce type de représentation, la connaissance du système sera facilement accessible à l'expert qui pourra ainsi aisément la modifier, l'agrandir ou encore simplement la consulter.

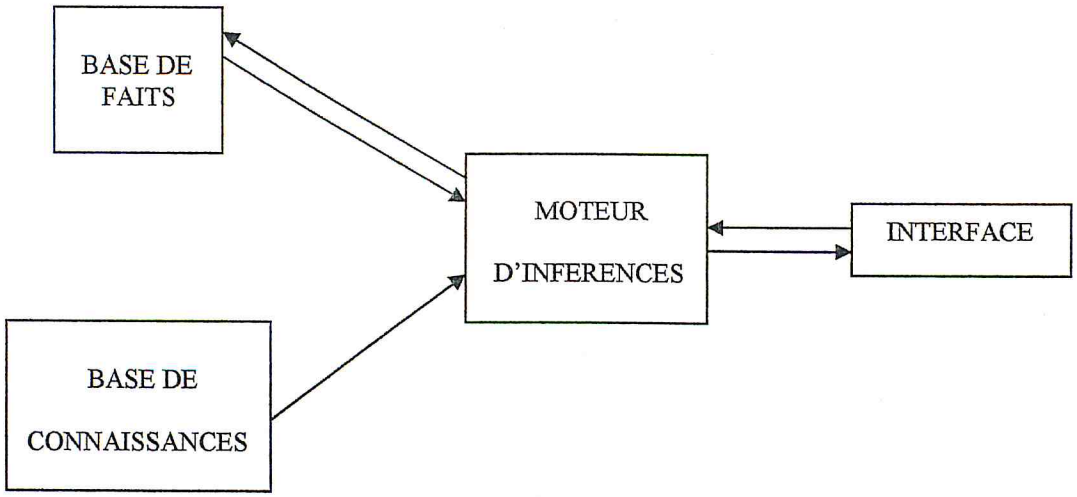


Figure I.3 - Architecture d'un système expert

3.2.1.1. Faits

Ce sont des éléments de type déclaratif qui constituent les atomes de la connaissance. Ce sont généralement des assertions dans un domaine précis.

Exemples :

Il fait beau
Age > 18
Taille = 1.85

3.2.1.2. Règles

Les règles représentent le savoir-faire de l'expert du domaine considéré.
Elles indiquent quelles conclusions tirer lorsque telles conditions sont satisfaites.
L'ensemble des règles constitue ce qu'on appelle la **base de règles (BR).** Cette dernière est en général *fortement structurée*. Le problème de cette structuration relève de ce qu'on appelle la *représentation des connaissances* et sera abordé en I.4.

Une règle est représentée sous forme de couple qu'on peut schématiser par (<condition>,<conclusion>), ou encore par la représentation :

Si {Condition} **alors** {Conclusion}

Exemple :

Si la voiture ne démarre pas
et les phares ne s'allument pas
et le klaxon ne fonctionne pas
Alors considérer un problème de batterie

3.2.1.3. Métaconnaissances

Pour qu'un système expert puisse modéliser un raisonnement humain, il est indispensable qu'il puisse raisonner sur ses propres raisonnements, réfléchir aux faits qu'il manipule, aux formules qu'il peut construire. Autrement dit, il n'est pas suffisant que le système ait des connaissances, il faut aussi qu'il ait des **métaconnaissances**.

De même que dans un système expert, la connaissance est traduite en règles, la métaconnaissance s'exprime aussi par des métarègles, c'est-à-dire des règles sur la manière d'utiliser les règles.

Exemple : On trouve dans MYCIN (système expert de diagnostic médical) la métarègle suivante:

Si on recherche une thérapie ***Alors*** considérer dans cet ordre les règles qui permettent de :

- acquérir les informations cliniques sur le patient
- trouver quels organismes, s'il en existe, sont causes de l'infection
- identifier les organismes les plus vraisemblables
- trouver tous les médicaments potentiellement utiles
- choisir les plus adaptés en plus petit nombre

3.2.2. Base de faits

La base de faits est la **mémoire de travail** du système expert. Elle est variable au cours de l'exécution et vidée lorsque l'exécution se termine. Au début de la session, elle contient ce que l'on sait du cas examiné avant toute intervention du moteur d'inférences. Puis elle est complétée par les faits déduits par le moteur ou demandés à l'utilisateur.

Par exemple, dans le domaine médical, la base de faits pourra contenir une liste de symptômes en début de session et un diagnostic lorsque celle-ci se terminera.

3.2.3. Moteur d'inférences

Le moteur d'inférences est la composante dynamique d'un système à base de connaissances : c'est un mécanisme qui permet d'inférer de nouvelles connaissances à partir de la base de connaissances du système.

Pour une situation donnée, il détecte les connaissances intéressantes, les utilise et les enchaîne. Il exécute ainsi les inférences au cours du processus de résolution, soit par modification, soit par adjonction d'éléments à la base de faits.

En résumé, le moteur d'inférences a pour rôle de simuler la réflexion de l'expert.

Un moteur d'inférences est généralement décrit par son type et son mode de fonctionnement.

3.2.3.1. Spécification d'un moteur d'inférences

Il existe plusieurs sortes de moteurs d'inférences. Ils sont spécifiés par un certain **ordre** qui détermine leur capacité d'abstraction et la nature des expressions qu'ils manipulent:

- ordre 0 : correspond à la logique des propositions. Les faits sont booléens, les formules sont de la forme: *fait* ou ¬ *fait.*
- ordre 0+ : correspond à la logique des propositions typées : (*attribut, opérateur, valeur)*
- ordre 1 : correspond à la logique des prédicats; possibilité de manipuler des variables.
  pour tout x, **si** distance(x) > 50 Km **alors** prendre un taxi.
- ordre 2 : correspond à la logique d'ordre 2.
  pour tout R, X, Y: **si** type(R)=symétrique **et** R(X, Y) **alors** R(Y, X)

3.2.3.2. Mode de fonctionnement d'un moteur d'inférences

Les règles peuvent être reliées de différentes façons. On parle alors de chaînage. Ce travail est attribué au moteur d'inférences. On distingue essentiellement trois principaux modes de fonctionnement:

- le chaînage avant,
- le chaînage arrière,
- et le chaînage mixte.

a) Chaînage avant

Le mécanisme du chaînage avant est très simple : pour déduire un fait particulier, on déclenche les règles dont les prémisses sont connues jusqu'à ce que le fait à déduire soit également connu ou qu'aucune règle ne soit plus déclenchable.

b) Chaînage arrière

Le mécanisme de chaînage arrière consiste à partir du fait que l'on souhaite établir, à rechercher toutes les règles qui concluent sur ce fait, à établir la liste des faits qu'il suffit de prouver pour qu'elles puissent se déclencher puis à appliquer *récursivement* le même mécanisme aux faits contenus dans ces listes jusqu'à ce que le but recherché soit atteint ou qu'aucune règle ne soit plus déclenchable.

c) Chaînage mixte

L'algorithme de *chaînage mixte* combine, comme son nom l'indique, les algorithmes de chaînage avant et de chaînage arrière.

3.2.3.3. Principe de fonctionnement

La figure suivante décrit le principe de fonctionnement d'un moteur d'inférences, ici basé principalement sur des systèmes utilisant des règles de production.

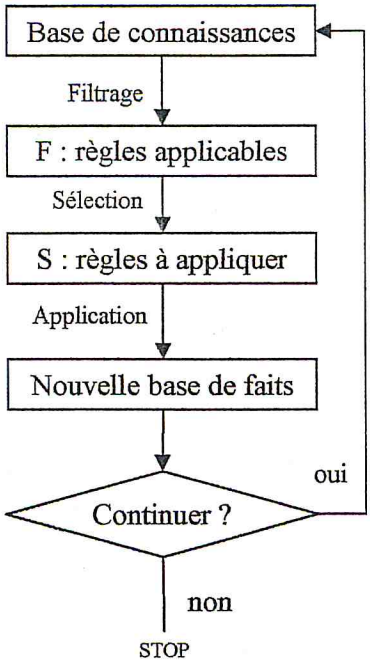


Figure I.4 - Principe de fonctionnement d'un moteur d'inférences

La figure ci-dessus met en relief les différentes étapes suivantes :

1. **Filtrage** : construction d'un ensemble F ne contenant que les règles dont les conditions de déclenchement ont été jugées satisfaites. F est appelé *ensemble de conflit.*

2. **Sélection** : appelée aussi *résolution de conflits* ; dans cette étape, le moteur détermine les règles, soit un sous-ensemble S de F, qui doivent être effectivement *déclenchées* (dans la majorité des cas S = F).

3. **Application** : application des règles de S et mise à jour de la base de faits en fonction de la partie conclusion de la règle appliquée.

4. **Test de continuation**:
   Si S est vide ou la base de faits correspond à l'état initial **alors** le système s'arrête.
   Sinon aller à 1
   Fin si

3.2.3.4. Stratégies de contrôle d'un moteur d'inférences

a) Système monotone / non monotone

Un système expert est dit *monotone* si et seulement si aucun fait ne peut être supprimé de la base de faits, et aucun fait ajouté n'introduit de contradictions dans la base de faits. La monotonie est une caractéristique souhaitable pour de nombreux systèmes de diagnostic car on suppose que l'état du sujet n'évolue pas au cours d'une session, par exemple. Un malade

présente généralement les mêmes symptômes au début et à la fin de la consultation chez son médecin.

Par contre, un système non monotone, permet de remettre en cause (supprimer ou modifier) des faits établis lors de la prise en compte de nouvelles connaissances. Cela est systématiquement le cas lorsque le système sert à piloter un processus physique, un robot par exemple.

b) Système à régime irrévocable / par tentative

On dit qu'un moteur fonctionne en mode irrévocable s'il considère que toutes les modifications apportées à la base de faits, restent valides même si le raisonnement aboutit à un échec. Inversement, le fonctionnement par tentative retire, en cas d'échec, tous les faits établis dans ce raisonnement et revient sur la résolution de conflits antérieure. Remettre en cause le déclenchement d'une règle et savoir quel sort réserver aux faits déduits lors de cette tentative infructueuse constitue un problème délicat. Une stratégie par tentatives est utile lorsque l'on veut établir des plans d'actions réclamant des actions irréversibles, mélange de deux liquides par un robot, par exemple. Pour établir de tels plans, il est clair que le système devra agir par tentatives car, si le mélange est effectivement réalisé et aboutit plus tard à un échec, il n'est plus possible de retrouver l'état initial.

Si les actions demandées ne sont pas irréversibles, une stratégie irrévocable sera plus appropriée et plus simple à mettre en œuvre.

c) Hypothèse du monde Clos / Ouvert

L'hypothèse du monde clos considère que toutes les assertions qui ne figurent pas dans la base de faits sont considérées comme « fausses », l'absence d'un fait dans la base de faits étant équivalent à sa négation. Par contre, l'hypothèse de monde ouvert considère tout fait n'appartenant pas à la base de faits comme « inconnu » (ni vrai ni faux). Dans ce cas, le démonstrateur dialogue avec l'utilisateur pour confirmation ou infirmation du nouveau fait.

3.2.4. Interface

Parfois non décrites comme entités propre au système lui-même, les interfaces utilisateurs sont pourtant indispensables pour son exploitation. Elles rendent la consultation du système facile et conviviale. Afin d'assurer ces fonctions avec efficacité, il faut que le dialogue entre l'homme et la machine soit aussi sophistiqué que possible.

Plusieurs possibilités sont envisageables pour réaliser cette communication :

- Rassembler dans un menu toutes les requêtes possibles qui pourraient être présentées par l'utilisateur, le système propose une panoplie de requêtes ; l'utilisateur n'a plus qu'à choisir l'une d'entre elles.
- Obliger l'utilisateur à apprendre un formalisme que reconnaît le système.
- Concevoir une interface permettant au système et à l'usager de communiquer en langage naturel.

Exemple : Dans le cas du chaînage avant, l'interface peut être un formulaire. Dans le cas du chaînage arrière ou mixte, elle peut être un module de questions posées à l'utilisateur.

Pour certaines applications, telles que les systèmes d'assistance, les systèmes pédagogiques ou les systèmes dont les connaissances sont évolutives, l'architecture de base est augmentée par d'autres modules :

A. Module d'explication

Un autre aspect fondamental de la technique des systèmes experts est la faculté d'incorporer dans les programmes des procédures d'explications de leur ligne de raisonnement. Le fait de disposer d'une représentation des connaissances sous une forme « qui ressemble » à celle des experts humains (par exemple : si le patient a de la fièvre et les jambes molles, alors il se pourrait qu'il ait la grippe) permet aisément à de tels programmes de montrer les inférences qu'ils ont faites successivement pour aboutir aux conclusions qu'ils donnent. Ceci est une qualité fondamentale même si le niveau d'explication qu'ils fournissent est encore peu profond ; les utilisateurs des systèmes experts pourront ainsi justifier leur confiance (ou leur défiance) à l'égard du programme, du fait que l'ordinateur cesse d'être une « boîte noire » comme trop souvent dans la programmation classique [Bonnet 84].

Grâce au module d'explication le système peut par exemple répondre à des questions du type « pourquoi as-tu besoin de telle information ? » dans le cas où l'utilisateur ne comprend pas les raisons de la question qui lui a été posée, ou du type « comment as-tu déterminé cette solution ? » dans le cas où l'utilisateur souhaite savoir comment un fait a été déterminé, ou comment une règle a été utilisée.

Le module d'explication peut aussi servir au concepteur du système en l'aidant notamment à l'élaboration, la validation et la maintenance de sa base de connaissances.

Le module d'explication est donc d'une utilité certaine, il permet :

- A l'informaticien, de l'utiliser comme mode de débogage afin d'affiner la démarche du moteur d'inférences.
- Au cogniticien, de vérifier la cohérence de la base de connaissances.
- A l'utilisateur final de l'utiliser si ce dernier a un doute sur la validité du diagnostic, ou s'il ne le comprend pas.

B. Module d'acquisition

Ce module permet d'acquérir les connaissances fournies par l'expert du domaine ou l'ingénieur de connaissances (cogniticien), et de gérer la cohérence de la base de connaissances suite à des mises à jour éventuelles de la part d'un expert.

Ce module permet entre autres :

- de faciliter l'ajout ou la modification des règles
- de tester la base de connaissances à tout instant
- d'accéder à toutes les informations contenues dans la base de connaissances.
- de réaliser le codage interne des règles dont il assure la maintenance

4. Techniques de représentation des connaissances

Le problème de la représentation des connaissances est celui de leur **transcription sous une forme symbolique** afin d'être exploitée par un système de raisonnement.

Un mode de représentation associe ainsi deux aspects imbriqués, voire confondus :

- la structure de données pour représenter l'information,
- la méthode associée d'exploitation de cette information, ou de raisonnement. Le mécanisme de raisonnement permet de découvrir dynamiquement de nouvelles informations et connaissances sur le problème traité. Cela distingue nettement une *base de connaissances* d'une base de données classique dont on ne peut extraire que les informations qui y ont été explicitement rangées.

Il existe deux grands types de **représentations de connaissances**, le plus souvent complémentaires :

- les représentations *procédurales*, qui contiennent la façon dont les connaissances codées doivent être utilisées.
- les représentations *déclaratives*, dans lesquelles les connaissances sont décrites indépendamment de leur exploitation ultérieure. Elles sont à l'origine de la programmation déclarative d'un système à bases de connaissances.

4.1. Représentations procédurales

On classe habituellement sous le terme de *représentation procédurale* l'introduction directe de procédures algorithmiques classiques dans les systèmes d'IA.

La représentation procédurale exprime par essence un flot d'information, elle traduit syntaxiquement comment transite l'information [Lauriere 82].

a) Les automates

Les états sont représentatifs de situations et les arcs représentent les règles de décisions pour passer d'un état à un autre.

b) Les procédures

Ce sont des programmes classiques désignés explicitement pour pouvoir les invoquer sans possibilité d'alternative.

Le déclenchement de ces procédures peut se faire de différentes manières :

- Par appel direct au moment où la procédure doit entrer en jeu ;
- Grâce à l'attachement procédural (c'est le cas des « frames ») où l'affectation d'une valeur à un attribut peut conduire au déclenchement d'une procédure associée ;
- Par action d'un *démon*, chargé de surveiller si une certaine condition est vérifiée et de déclencher, le cas échéant, la procédure spéciale correspondante. On traite

ainsi des cas particuliers qui sortent des procédures générales et qui alourdiraient le programme, mais la surveillance permanente est d'utilisation coûteuse ;
- Par appel dirigé par un schéma spécifiant le but à réaliser. La procédure dont le schéma caractéristique coïncide avec le schéma correspondant à la tâche qui lui est assignée est déclenchée.

Quel que soit le mode de représentation adopté, il faut toujours un mécanisme pour interpréter la connaissance et l'« exécuter ». C'est le cas par exemple du moteur d'inférences d'un système à règles de production.

Par ailleurs, les métaconnaissances et les connaissances de nature heuristique s'expriment souvent plus facilement de façon procédurale.

Enfin, l'attachement procédural permet une représentation mixte mieux appropriée à la traduction de la connaissance et du raisonnement humain.

La représentation procédurale perd en modularité et en souplesse. Elle implique la connaissance *a priori* de la façon de résoudre le problème posé et enfin ne favorise pas la mise en oeuvre d'un mécanisme d'explication, ni plus que la génération de plans d'actions, ce qui laisse peu de place à l'adaptabilité du système.

4.2. Représentations déclaratives

Ce sont des connaissances données en vrac, de façon totalement modulaire. Elles sont indépendantes des programmes d'utilisation.

Les connaissances déclaratives sont inutilisables sans connaissances procédurales. En effet, une connaissance déclarative en tant que tel n'a aucune valeur sémantique, c'est un interpréteur (programme) qui exprime la façon dont celles-ci va être utilisée [Voyer 87].

4.2.1. Représentations logiques

Issues des développements théoriques dans le domaine de la logique formelle, ces représentations remontent aux premiers jours de l'intelligence artificielle (Cf. le *Logic Theorist* de Newell, Shaw et Simon en 1956). Archétype de la représentation déclarative, elles concernent surtout la logique mathématique (logique des propositions et logique des prédicats du premier ordre) ; mais d'autres logiques non standard sont également utilisées (multivaluées, modales, etc.)

a) La logique des propositions

La logique des propositions d'ordre 0

En logique d'ordre 0, une action (ou un état) recouvre l'idée d'assertion formulée suivant une certaine syntaxe et susceptible de prendre dans un univers donné la valeur « vrai » ou « faux ».

Exemple : « Il pleut à Alger ».

De telles assertions élémentaires peuvent être niées par l'opérateur « négation » (symbolisé par ¬) et associées entre elles grâce aux connecteurs logiques ET (∧), OU (∨), l'implication logique (→) ou encore l'équivalence (↔), pour former des formules logiques composées appelées aussi formules bien formées (FBF).

La logique avec variables globales (d'ordre 0+)

En logique d'ordre 0+, une action peut être constituée de test sur des objets précis. Elle sera généralement représentée sous forme de triplet <objet , comparateur , valeur>.

Exemple : Température > 37.5

La logique des propositions se révèle insuffisante lorsque l'on veut déduire des propriétés valables pour des ensembles d'éléments du monde réel. Ainsi, exprimer que tous les hommes sont mortels ne peut se faire qu'en particularisant tous les hommes et en énonçant que chacun d'eux est mortel.

La logique des prédicats du premier ordre fournit les moyens de préciser la portée d'assertions plus générales et donc d'exprimer ce type de connaissance.

La logique des prédicats du premier ordre peut être vue comme une extension de la logique des propositions. Munie des propriétés de base de la logique des propositions, elle permet en outre d'introduire des éléments généraux appelés « variables ». Ces variables habituellement notées u, v, …, z, peuvent être quantifiées par le quantificateur universel ∀ (quel que soit) ou le quantificateur existentiel ∃ (il existe).

Ainsi, $(\exists\ x)\ P(x)$ signifie qu'il existe au moins un élément (ou individu) x du domaine pour lequel $P(x)$ est vrai.

Par ailleurs, $(\forall y)\ Q(y)$ signifie que, pour tous les y du domaine de définition de Q, Q(y) est vrai.

Exemples :

$(\forall x)\ (\forall y)\ (\forall z)\ ((PERE(x,y) \wedge PERE(x,z)) \rightarrow FRERE(y,z))$
$(\exists x)\ (FRUIT(x) \wedge COULEUR(x, JAUNE))$

Un intérêt majeur des représentations logiques réside dans la disponibilité de règles de raisonnement fondées sur ces formules : *modus ponens*, *principe de résolution*, *modus tollens*, etc...

La logique formelle est fondée sur de solides bases théoriques et représente dans de nombreux cas la façon naturelle d'exprimer des faits et des relations déductives entre les faits. Elle est parfaitement adaptée au raisonnement exact disposant de données complètes, comme la démonstration de théorèmes, la résolution d'énigmes, etc.

Cependant, la rigueur de ce formalisme, dans lequel les prédicats prennent les valeurs binaires VRAI ou FAUX, ne permet pas d'exprimer des appréciations nuancées. Par ailleurs, il ne permet pas de prendre des décisions dans le cas d'informations manquantes, contrairement à la capacité de l'homme à faire du raisonnement par défaut. Pour pallier ces

limitations, d'autres logiques ont été introduites, notamment les logiques non monotones, les logiques multivaluées et les logiques modales.

Par ailleurs, la logique formelle ne fournit pas de moyens d'organisation de l'ensemble des connaissances manipulées, ce qui, dans le cas où cet ensemble est important, présente un inconvénient lors de son exploitation. Enfin, représenter des connaissances de nature procédurale ou heuristique est assez difficile.

4.2.2. Réseaux sémantiques

Issus des travaux de psychologie cognitive sur l'organisation de la mémoire en réseaux associatifs, les réseaux sémantiques ont été à l'origine appliqués à l'interprétation du langage naturel.

Les réseaux sémantiques ou conceptuels permettent de représenter les liens qui unissent un concept avec d'autres. Ce sont des graphes étiquetés où chaque concept est représenté par un noeud et chaque lien forme un arc, étiqueté du nom de la relation entre les deux concepts, éventuellement pondéré pour donner des priorités ou des coûts à certains chemins dans le réseau. Le sens d'un réseau sémantique provient de l'interprétation des liens. Une telle architecture permet d'inférer de nouvelles relations entre concepts par héritage ou par propagation des propriétés d'un concept à travers les liens du réseau.

Ils permettent notamment d'exprimer *une hiérarchie* ou *une taxinomie* de concepts et la proximité entre des connaissances. Les réseaux sémantiques constituent donc *un moyen de structuration* des bases de connaissances, problème important lorsque celles-ci sont de grande taille.

Par exemple, PROSPECTOR utilise une taxinomie sur les minéraux (figure I.5), qui lui permet de savoir que les pyrites sont des sulfures et des produits d'altération. Les arcs marqués « e » indiquent que le nœud est un élément de l'ensemble père, les arcs marqués « s » signifient un sous-ensemble, tandis que les arcs « de » indiquent des éléments différents d'un même père.

Les réseaux sémantiques sont bien adaptés à la représentation d'un ensemble hiérarchique de concepts. Le mécanisme d'héritage de propriétés permet une forme grossière de raisonnement. Les limitations à ce niveau font des réseaux un mode de représentation plutôt utilisé en complément d'autres modes. Cependant, les langages hybrides de représentation, du type KL-ONE, permettent une intégration parfaite avec une logique du premier ordre. On peut ainsi disposer d'un système de représentation des données tout en conservant l'expressivité de la logique et les riches possibilités de raisonnement qu'elle permet via un moteur d'inférences.

Par contre, malgré des tentatives de normalisation, il n'existe pas actuellement en la matière de terminologie standard, notamment dans les types de relations. Enfin, il n'est pas possible d'exprimer des connaissances de nature procédurale. Cette limitation n'existe plus dans les réseaux d'objets, étudiés par la suite.

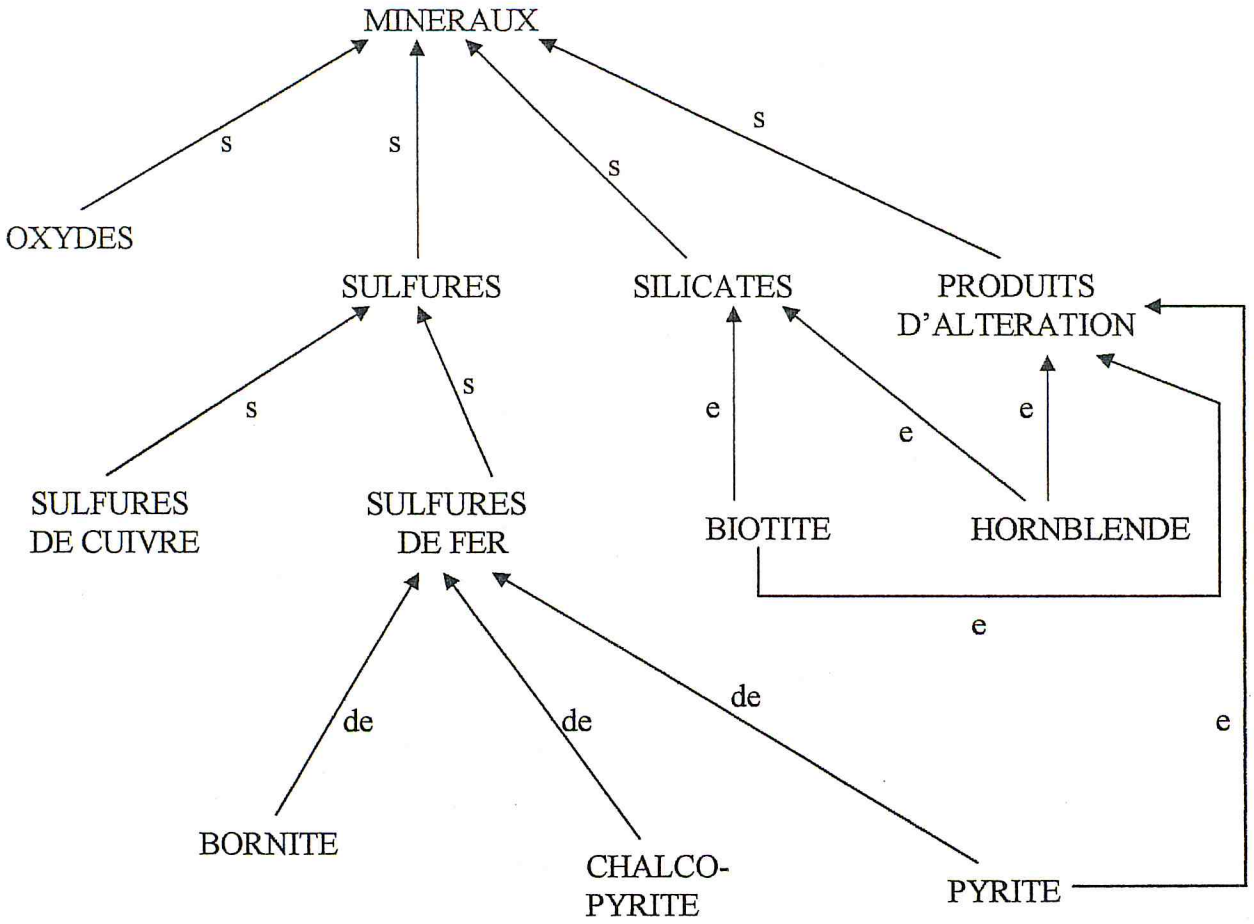


Figure I.5 - Une taxinomie dans PROSPECTOR (tirée de [Benchimol et al 90])

4.2.3. Règles de production

Avec l'avènement des systèmes experts, les règles de production ont acquis une grande popularité en IA, notamment grâce au système de diagnostic médical MYCIN. Une règle de production est un quantum de connaissance, déclaratif et autonome, de la forme :

SI **conditions** *ALORS* **conclusions (coefficient)**

qui signifie que, si les prémisses (partie conditions) sont vérifiées, on peut tirer des conclusions (ou déclencher une action, une procédure de calcul ou d'affichage, etc.).

Dans cet exemple, la règle est accompagnée d'un coefficient qui traduit la confiance qui lui est accordée ou encore la vraisemblance de cette connaissance.

Les systèmes experts ont pour la plupart une partie au moins de leur base de connaissances représentée dans ce formalisme.

Les coefficients affectés aux règles permettent de faire du raisonnement « incertain » et résultent le plus souvent de l'expérience de l'expert qui est à l'origine de la base de connaissances.

On trouve par exemple, dans MYCIN la règle suivante:

Si le lieu du prélèvement est le sang
et la morphologie de l'organisme est bâtonnet
et l'organisme est à gram négatif
et le patient a un organisme fragilisé
Alors il est probable (0.6) que l'organisme soit le Pseudomonas-aeruginosa

Les règles sont des morceaux de connaissance indépendants qui ne se réfèrent pas les uns aux autres.

Chaque règle contient dans sa partie prémisses les conditions de son application. Elles constituent par conséquent un moyen simple et naturel de traduire la connaissance heuristique mise enjeu dans un raisonnement de type conditionnel et sont fréquemment utilisées, notamment dans les systèmes experts.

De plus, le cheminement suivi par le moteur d'inférences peut être explicité de façon à fournir à l'utilisateur des explications sur le raisonnement suivi par le système.

Enfin, l'introduction de coefficients permet de pondérer les jugements exprimés dans les règles, de prendre en compte des données imprécises et de traiter des problèmes qui ne relèvent pas d'une logique binaire pure.

Par ailleurs, si les règles représentent bien la connaissance heuristique « de surface » utilisée par un expert, elles ne permettent pas de rendre compte des connaissances sur les concepts « profonds » sur lesquels s'appuie l'expert.

En conséquence, si les systèmes à bases de connaissances renferment la plupart du temps des règles de production, on associe souvent aux règles d'autres modes de représentation.

4.2.4. Les frames et les objets structurés

La notion de *schéma* a été proposée comme modèle de représentation prototypique d'expériences passées mises à profit pour résoudre un problème nouveau. Ce concept a été repris en intelligence artificielle sous la forme de *frames*, proposés en 1974 par M. Minsky en vision par ordinateur et de *scripts* ou *scénarios*, introduits en 1977 par R. Schank pour la compréhension du langage naturel.

Plus récemment, ces formalismes ont été intégrés dans le concept plus général de « représentation par objets », notamment à la suite des travaux sur les langages de programmation à objets.

a) Frames et scénarios

Un frame est une entité regroupant de façon structurée l'ensemble des connaissances relatives à un objet, un concept ou une situation typique.

Le concept de CADRE pourrait se décrire ainsi :

```
< frame CADRE
  sorte de : EMPLOYE
  nom : (nom-de-famille, prénom)
  date-de-naissance : (jour, mois, année)
  âge : (années)
        (si besoin calculer l'âge)
  service : (valeurs possibles : commercial / administration)
  date-de-début : (mois, année)
  jusqu'à : (mois, année)
            (par défaut : maintenant)
  ...
>
```

Un frame se compose donc d'un ensemble d'attributs ou de cases (*slots*), comme « nom » ou « service », correspondant à autant de notions associées à ce concept. Chaque attribut peut être décrit selon différentes facettes, par exemple « valeurs possibles », « défaut », etc. Cet exemple illustre les propriétés essentielles des *frames*, en particulier l'existence :

- de valeurs par défaut associées aux attributs. Ces valeurs permettront peut-être la poursuite du raisonnement jusqu'à ce qu'une éventuelle remise en cause de ces valeurs intervienne (raisonnement par défaut),
- d'intervalles ou d'ensembles de valeurs possibles,
- de procédures, comme *si besoin* (la procédure se déclenche si le raisonnement réclame la valeur de l'attribut) ou *si echec* qui indique au système que faire s'il n'a pas trouvé de valeur pour un attribut. Il s'agit ici d'attachement procédural.

Les scénarios reposent sur le même principe de représentation mais ils sont relatifs à des séquences typiques d'événements et non plus à des descriptions statiques d'objets.

b) Les objets structurés

Dans le formalisme des objets, l'univers d'une application se compose d'un ensemble hiérarchique de *classes* génériques et d'*instances d'objets*, muni du mécanisme *d'héritage de propriétés*. Chaque objet est décrit, comme un frame, par des attributs. Il comprend un ensemble de *méthodes*, procédures assurant la manipulation des informations encapsulées dans l'objet.

On peut de même attacher à un objet, ou à un de ses attributs, un ensemble de règles de production chargées de résoudre un sous-problème.

Les objets d'un univers sont indépendants et communiquent entre eux par un mécanisme de *transmission de messages*. Un message envoyé à un objet spécifie un certain traitement assuré par une des méthodes de l'objet.

L'exemple suivant décrit l'objet « cercle » :

(Classe : cercle

Attributs : rayon
abscisse
ordonnée

Méthodes : périmètre : retourner (2 * pi * rayon)
surface : retourner (pi * rayon^2))

Les représentations par objets sont actuellement largement utilisées car elles présentent de nombreux avantages :

- structuration d'un univers d'application, permettant éventuellement une modélisation de cet univers ;
- rassemblement de l'ensemble des éléments de connaissances relatifs à un concept (par exemple, en médecine, tous les aspects d'une maladie : symptômes, signes, maladies associées, thérapeutiques, séquelles, etc.) ;
- couplage des aspects déclaratifs et procéduraux ;
- hiérarchisation des connaissances et des raisonnements.

5. Applications des systèmes experts

5.1. Apports d'un système expert

Les fonctions assurées par les systèmes experts sont nombreuses ; nous citons entre autres [Benchimol et al 90]:

– *l'interprétation* : traduction de signaux, provenant de capteurs par exemple, ou de données brutes , en expressions symboliques pouvant être utilisées dans des raisonnements ;

– *le diagnostic* : établissement d'une corrélation entre des caractéristiques ou symptômes des situations types ;

– *la formation* : transmission de connaissances à un élève dont le niveau et les caractéristiques ont fait l'objet d'un diagnostic en vue de l'enseignement le mieux adapté ; cette transmission du savoir ou de savoir-faire peut porter sur le diagnostic, la maintenance, la conception, etc. ;

– *la surveillance* : déclenchement d'une alarme dans des conditions déterminées pouvant évoluer avec le contexte ou envoi d'un compte rendu à partir de signaux interprétés et utilisés dans un diagnostic ;

– *la prévision* : description d'une situation par anticipation, à partir d'une situation courante, généralement au moyen d'un modèle construit sur une base historique ou par apprentissage ;

– *la simulation* : déduction, à partir d'un modèle, des conséquences d'actions ou d'événements déclenchés par le système lui-même au cours du déroulement de la simulation ;

– *la planification* : définition dans le temps et dans l'espace, des actions permettant d'atteindre un état final, en comparant l'état courant à l'état souhaité et en prévoyant les conséquences des actions, de manière à respecter les contraintes imposées par l'environnement, le niveau des ressources disponibles et les conséquences prévisibles des interactions entre états et actions ou entre états successifs ;

– *la maintenance* : plan d'action particulier découlant d'un diagnostic mettant en lumière les défaillances d'un système et identifiant leurs causes ; en mode interactif, la simulation permet de confronter les résultats des tests avec ceux que donnerait un système en ordre de marche ; le plan d'action consiste à fournir les instructions nécessaires pour effectuer la réparation ;

– *la conception* : ensemble de choix de décisions permettant, à partir de performances éventuellement fixées à la suite d'un diagnostic, de déterminer le cahier des charges d'un but à satisfaire, en fonction de besoins exprimés à un instant donné, et de donner les moyens d'atteindre cet objectif, en définissant les spécifications d'un produit ou d'un processus qui respectent le cahier des charges ;

– *le contrôle et le pilotage* : ensemble d'actions appliquées à un système d'après les informations résultant de la surveillance du système et de l'anticipation des situations à venir en vue d'assurer, par une maintenance permanente et une réponse adaptée aux divers aléas, un fonctionnement du système se rapprochant le plus possible du fonctionnement normal défini par des valeurs de consigne et le programme résultant d'une planification adéquate.

5.2 Exemples de systèmes experts

5.2.1. Systèmes experts spécifiques :

De nombreux systèmes experts ont été conçus, depuis les années 70, et dans divers domaines.

5.2.1.1. MYCIN [Alty et al 86]

MYCIN a été développé à l'université de Stanford pour assister les médecins avec des conseils pour le diagnostic et le traitement des maladies infectieuses. Il a donc pour but d'identifier l'organisme qui est à l'origine de la maladie et de proposer un traitement.

Les faits sont enregistrés sous la forme de triplets CONTEXTE-PARAMETRE-VALEUR. Un contexte est une entité du monde réel, un patient par exemple. Un paramètre est un attribut du contexte (l'âge par exemple) et la valeur est une occurrence du paramètre (25 ans par exemple). A chaque triplet du fait est associé un facteur de certitude (FC) dont la valeur est comprise entre -1 (négation) et +1 (certitude). Par exemple :

ORGANISME1-IDENTITE-PSEUDOMONAS 0.8
signifie que « l'identité de l'organisme 1 est pseudomonas, avec un facteur de certitude de 0.8 ».

Le raisonnement sur le domaine des connaissances est fait grâce au codage du savoir de l'expert sous la forme d'un ensemble de règles de production du type :

SI prémisse ALORS action (FC)

où la prémisse est une conjonction de triplets et où l'action correspond en général à l'instanciation d'un triplet. Le facteur de certitude FC de la règle est utilisé en liaison avec les FCs des triplets concernés, pour calculer le nouveau facteur de certitude du triplet résultant de l'action.

Voici un exemple de règle de MYCIN :

SI 1) on sait que l'organisme a pu se développer de manière aérobique

ET 2) le site de la culture est le sang
OU le laboratoire a essayé de faire croître l'organisme anérobiquement

ET 3) l'organisme a pu se développer anaérobiquement

ALORS On peut penser que l'aérobicité de l'organisme est soit facultative (0.5), soit anaérobique (0.2)

En LISP, cela s'écrit :

```
Prémisse :  ($AND(NON CONNU CNTXT CROISSANCE-DANS-L'AIR)
                 ($OR(MEME CNTXT SITE-SANG)
                      (MEME CNTXT ESSAI-ANAEROBIQUE)
                 (MEME CNTXT SITE-SANG)

Action :         (CONCLURE CNTXT REACTION-AIR' ((FACULTATIVE 500)
                 (ANAEROBIQUE 200)))
```

Lorsqu'un système est fait pour aider les médecins dans le traitement d'un patient, le médecin utilisateur doit avoir confiance dans son raisonnement au cours d'une consultation. C'est pourquoi MYCIN (comme de nombreux autres systèmes experts) peut expliquer et justifier ses conclusions. Une telle faculté est utile aussi pour l'enseignement. A tout moment, l'utilisateur peut demander *COMMENT* et *POURQUOI* il a pris telle décision. L'exploitation de l'architecture à base de règles simplifie l'obtention de telles explications.

Quoique la capacité d'explication de MYCIN soit relativement importante, elle est limitée aux informations relatives aux environs immédiats de l'état du raisonnement, durant la consultation. De plus, MYCIN ignore de nombreux aspects des connaissances, par exemple, pourquoi Pseudomonas est une bactérie ? Il ne dispose d'aucune connaissance causale explicite du domaine considéré.

5.2.1.2. **PROSPECTOR** [Alty et al 86]

PROSPECTOR a été développé pour aider les géologues. Il fut conçu pour fournir trois grands types de conseils : évaluation de sites pour déterminer la probabilité d'existence de certains gisements ; évaluation des ressources géologiques d'une région, et sélection des sites les plus favorables à une exploitation donnée. Le programme a été écrit par SRI International en association avec les consultants géologues et l'organisme *U.S. Geological Survey.*

Les experts géologues utilisent des modèles très spécifiques pour identifier les zones susceptibles de contenir des gisements de minerai. Dans PROSPECTOR, ces modèles sont codés dans le système et systématiquement interprétés pour donner des conseils pour l'évaluation du site.

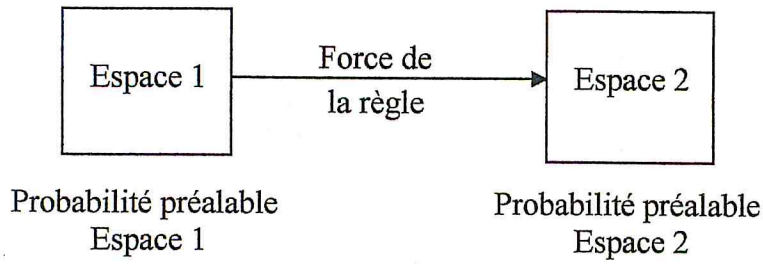


Figure I.6 - Détail d'un modèle de PROSPECTOR

Un modèle est constitué d'espaces reliés par des règles. Un espace peut être un indice observable ou une hypothèse et chacun possède une valeur de probabilité qui indique son degré de vérité. Les règles spécifient comment la modification de la probabilité d'un espace en affecte un autre. Ceci est illustré par la figure I.6. Ainsi, si la probabilité de départ de Espace 1 évolue, la règle en propage l'effet vers Espace 2.

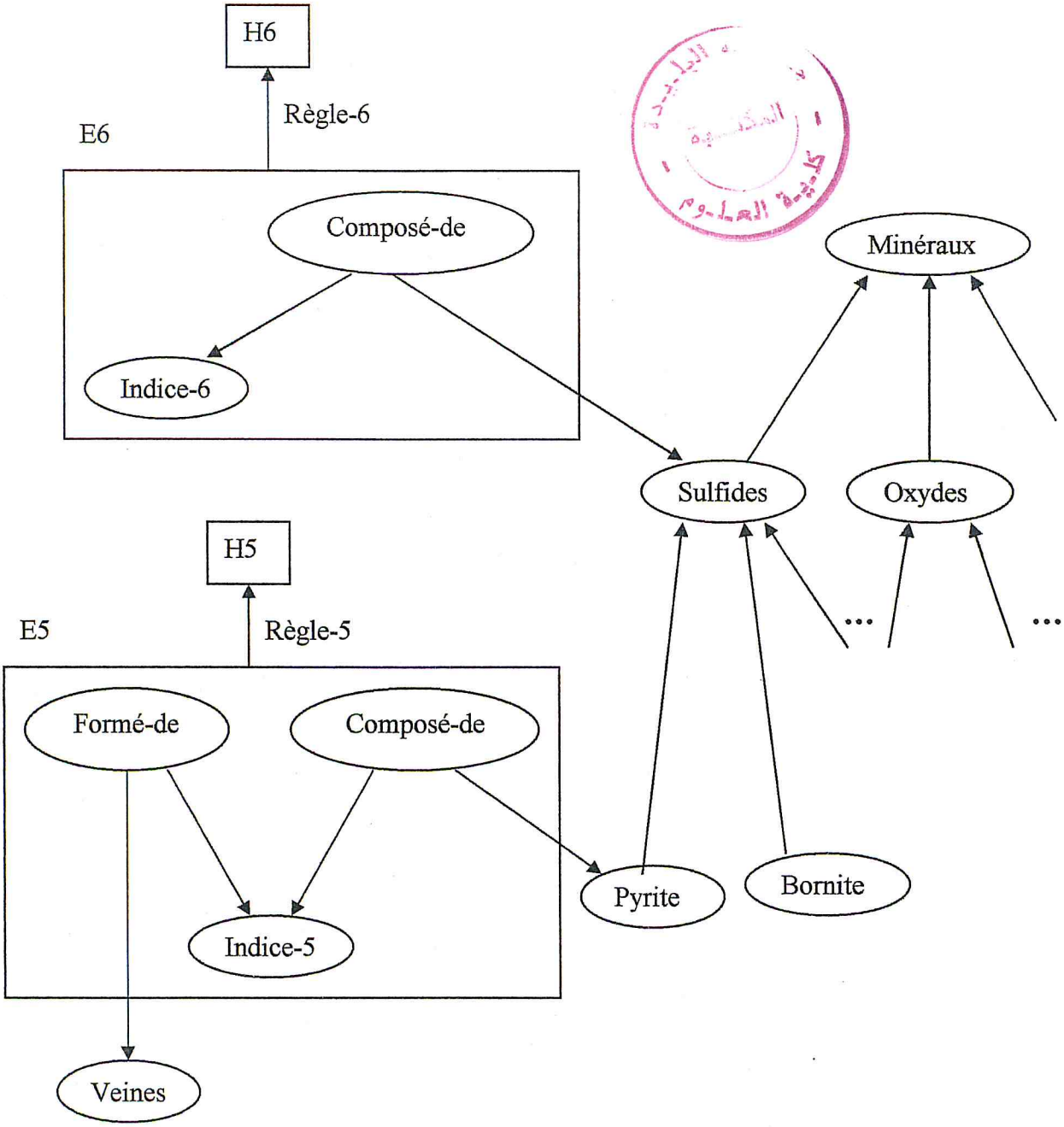


Figure I.7 – Partie du réseau sémantique de PROSPECTOR

L'analogie avec MYCIN est évidente. Espace 1 correspond à une prémisse et Espace 2 à une action. Un modèle est élaboré en reliant des espaces au moyen de règles, pour former un *réseau*. Notons que dans ce réseau, les règles peuvent être utilisées avec les connecteurs ET, OU et NON. Chaque fois que la probabilité d'un espace change, l'effet se propage à travers la réseau. La figure I.7 présente un exemple de réseau de PROSPECTOR (une partie d'un modèle).

5.2.1.3. R1 [Alty et al 86]

L'objectif de R1 est, à partir d'une liste de besoins liées au système d'un VAX-11, exprimés par un client, de calculer une configuration des composants, qui fonctionne et qui soit adaptée à la demande. Ces composants sont indiqués par le système, en sortie, sous forme d'un ensemble de diagrammes. Le processus mis en œuvre comporte trois activités interdépendantes : déterminer si la configuration est complète en remédiant aux manques éventuels, constituer une configuration viable, et produire un diagramme des relations spatiales appropriées entre les divers composants.

R1 décide si une configuration est cohérente, à partir de connaissances sur chacun des composants et sur les relations entre eux. Le VAX 11/780 par exemple, peut être configuré à partir de plus de 400 composants, chacun pouvant posséder jusqu'à 8 propriétés. La base de connaissances comporte par conséquent plus de 3000 informations sur les composants. Une organisation spatiale satisfaisante dépend non seulement de ces connaissances mais aussi d'autres notions liées à la géométrie des composants et des pièces qui les supportent.

Chaque entrée dans la base de connaissances contient une description d'un composant, comportant son nom et un ensemble de couples attribut/valeur. Chaque composant a un type et une classe, indiquant le type du composant (par exemple unité de disque) et la classe d'attributs qui lui sont lié (armoire, interface synchrone de fond de panier (SBI), etc...) dans la base de connaissances. La figure I.8 (extraite d'un des rapports sur R1) montre un ensemble caractéristiques d'entrées.

RK711-EA est un groupe de composants. Il contient un câble de 25 pieds (070-12292-25), une unité de disque (RK07-EA*) et un groupe de composants (RK611).

RK-711EA
Classe : Groupe
Type : Unité Disque
Disponible : Oui
Liste de composants : 1 070-12292-25
1 RK07-EA*
1 RK611

RK07-EA*
Classe : Unibus
Type : Unité disque
Disponible : Oui
Rang : 8
Profondeur : 28 inches
Largeur : 24 inches
Hauteur : 42 inches
Module unibus requis : RK611*
Ports : 1
Voltage : 120 volts
Fréquence : 60 hertz
Type de câble requis : 1 070-12292 depuis module Unibus Unité disque
ou 1 070-12292 depuis équipement Unibus Unité disque

RK611
Classe : Groupe
Type : Unité disque
Disponible : Oui
Liste de composants : 3 G727
1 M9202
1 070-12412-00
1 RK611*

Figure I.8 - Eléments de la base de connaissances

La figure suivante montre un exemple d'une règle de R1 :

Assignation –Alimentation-6

SI : le contexte courant actif est assignation alimentation électrique
ET un adaptateur Unibus a été mis dans une armoire
ET la position qu'il occupe (son Nexus) est connue
ET il reste de la place dans l'armoire pour une alimentation pour ce Nexus
ET il y a une alimentation disponible
ET il n'y a pas de régulateur H7101 disponible
ALORS : Ajouter un régulateur H7101 à la commande

Figure I.9 - Une règle de R1

La base de connaissances contient également un ensemble de gabarits qui permettent à R1 de déterminer quel est le taux de remplissage d'un élément à un moment donné, et à quel endroit placer un composant. Elle contient aussi des informations sur les contextes, les configurations partielles et le résultat de diverses sortes de calcul.

Le tableau suivant récapitule quelques systèmes experts dans divers domaines : [Bonnet 84]

Domaine	**Thème**	**Nom**	**Auteur(s)**
Médecine	-Infections du sang et méningite -Médecine interne -Maladies rénales -Infections pulmonaires -Cardiologie	-MYCIN -INTERNIST -PIP -PUFF -DIGITALIS	-Shortliffe -Pople -Pauker -Kunz -Gorry
Biologie	-Suggestion de plans d'expérience -Analyse de protéines	-MOLGEN -CRYSALIS	-Martin -Engelmore
Chimie	-Interprétation de données de spectrographe de masse -Avec apprentissage -Synthèse organique -Synthèse organique	-DENDRAL -META-DENDRAL -SECS -SYNCHEM	-Feigenbaum - Buchanan -Wipke -Gelenter
Physique	-Résolution de problèmes de mécanique -Analyse de circuits électriques -Analyse de circuits éléctriques -Résistance de matériaux	-MECHO -SOPHIE -PEACE -SACON	-Bundy -Brown -Dincbas -Langley
Géologie	-Minérale -Pétrolière -Pétrolière -Pétrolière	-PROSPECTOR -LITHO -DIPMETER ADVISOR -DRILLING ADVISOR	-Duda -Bonnet -Davis -Hollander
Mathématiques	-Découvert de concepts -Résolution d'intégrales, équations différentielles, etc... -Apprentissage de techniques de résolutions d'intégrales	-AM -MACSYMA -LEX	-Lenat -Moses -Mitchell
Ordinateurs	-Configuration de VAX -Diagnostic de pannes	-R1, XSEL, XCON -DART	-McDermott -Bennett
Programmation automatique	-Synthèse de programmes -Synthèse de programmes	-PECOS -DEDALUS	-Bartsow -Manna
Fabrication	-Conseils en conception de gammes d'usinage	-GARI	-Descotte
Militaire	-Interprétation de signaux	-HASP / SIAP	-Nii

Figure I.10 – Tableau récapitulatif de quelque Systèmes Experts

5.2.2. Systèmes experts généraux :

Un *système général* ou *générateur de systèmes expert*, ou encore *shell* est un outil spécialement conçu pour la génération de systèmes experts. Ceci implique qu'il dispose d'une série d'outils spécifiques au développement de systèmes experts : moteur d'inférences, interfaces utilisateurs, formalisme de représentation de la connaissance, gestion de la connaissance incertaine, aide au développement, etc...

Il va de soi que ce genre d'outils facilite le développement. Aussi, il n'est pas rare de constater que dans la pratique, on développe un premier prototype à l'aide d'un shell pour traduire ensuite en C la version définitive (pour des raisons d'efficacité).

5.2.2.1. EMYCIN

EMYCIN (Essential MYCIN) a été réalisé sous la direction de Van Melle (1979), il est issu du système expert MYCIN.
EMYCIN a été appliqué dans divers domaines et a permis de développer de nombreux systèmes experts parmi lesquels :

- PUFF : maladies pulmonaires
- HEADMED : psycho-pharmacologie
- SACON : construction mécanique
- DART : pannes d'ordinateurs
- SECOFOR : incidents de forages pétroliers
- LITHO : détermination de lithofaciès
- TOM : maladies de la tomate

5.2.2.2. OPS 5

OPS 5 est la dernière version développée par Charles Forgy (1981) de plusieurs langages OPS précédemment réalisés à l'université Carnegie-Mellon de Pittsburgh, en particulier par Forgy et McDermott (1977).
De même que EMYCIN, OPS 5 a donné lieu à de nombreux systèmes experts :

- R1 / XCON : configuration d'ordinateurs
- ACE : surveillance de câbles téléphoniques
- AIRPLAN : décollages et appontages d'avions
- AI-SPEAR : suivi d'ordinateurs
- YES / MVS : pupitrage d'ordinateurs

5.2.2.3. CLIPS

CLIPS (C Language Integrated Production System) est le résultat d'un projet de la NASA qui a débuté en 1984. Ce fut le premier générateur de systèmes experts écrit dans un langage évolué (le langage C).

CLIPS a reçu une large acceptation dans l'industrie et le milieu universitaire pour diverses raisons parmi lesquelles :

- portabilité : il existe des versions pour différentes plates-formes, PC (Windows, Dos), Unix, Macintosh.
- rapidité : il bénéficie de la vitesse d'exécution du langage C.
- faible coût : il peut être téléchargé gratuitement sur Internet.
- capacité d'extension : le code source est fourni gratuitement.
- documentation : elle est abondante et de nombreux ouvrages y sont consacrés.

6. Limites des systèmes experts de première génération

Alors qu'il a pu sembler initialement que l'approche préconisée dans les systèmes experts de première génération (SE1G) permettrait de développer des SBC plus rapidement et plus facilement, elle a en fait conduit à de nombreuses difficultés :

- Acquisition des connaissances
- Production d'explications
- Manque de robustesse
- Modification difficile des stratégies de contrôle
- Impossibilité de réutiliser tout ou partie de ces systèmes

6.1. Acquisition des connaissances

L'acquisition des connaissances s'est avérée être un goulot d'étranglement pour reprendre l'expression utilisée. Ceci est dû au fait que la construction d'un système expert de première génération est essentiellement un travail d'acquisition des connaissances. La préoccupation principale du cogniticien est de collecter les règles qui constitueront la base de connaissances de son système dans la perception qu'il peut avoir de son travail.

L'acquisition des connaissances constitue la tâche la plus importante, aussi bien quantitativement, de par le temps passé, que qualitativement, de par son importance sur le résultat final. Ce goulot d'étranglement est également dû au fait que les connaissances que l'on cherche à acquérir ne sont pas nécessairement, de par leur nature, les connaissances les plus faciles à éliciter pour les experts. Ceux-ci sont en général plus enclins à expliquer les principes du domaine, ou à rationaliser leurs comportements, qu'à fournir les associations heuristiques dont ont besoin de tels systèmes. Enfin, cette difficulté provient également du fait qu'aucun guide, qu'aucune méthode ne permet de structurer cette tâche. Rien ne permet d'aider l'expert et le cogniticien à identifier les connaissances qui devront être représentées dans le système ou à focaliser l'élicitalion.

L'acquisition des connaissances, constituera une motivation et un vecteur de progrès importants pour bon nombre de travaux sur les systèmes experts de deuxième génération (SE2G).

6.2. Explications

La capacité des systèmes experts de première génération à expliquer leur raisonnement ou à justifier leurs conclusions s'est vite avérée extrêmement limitée.

En effet, l'objectif prioritaire de leurs concepteurs était d'avoir des systèmes capables de fournir des solutions précises et exactes aux problèmes ; ils ne s'attachaient pas, en premier lieu, à ce que le système utilise pour cela des méthodes similaires à celles des experts humains.

Bien qu'il fût souhaitable que le système puisse expliquer son raisonnement, il n'était pas conçu en fonction de cet objectif et son optimisation portait plus sur l'efficacité du raisonnement que sur sa cohérence. Les *outils d'explication* tendaient plus à répondre aux

questions sur le processus abstrait des inférences utilisées par le système qu'à essayer d'expliquer le raisonnement avec des références directes au domaine du problème. De plus, ces outils n'étaient pas conçus pour faire face au problème de l'*adaptation* de l'explication du raisonnement aux divers utilisateurs, dont les connaissances et la compréhension des problèmes varient de l'un à l'autre.

Une première limitation est due en partie à la nature des connaissances représentées qui limitent l'explication à une trace des règles utilisées. Les connaissances du domaine étant implicites dans la base de connaissances, elles restent inaccessibles au module d'explication. Il s'agit là d'un manque de représentation *profonde* des associations phénoménologiques qu'ils sont capables de faire. Même si un programme comme MYCIN est très puissant pour relier symptômes et diagnostics, interpréter des cultures, etc..., il n'a aucun modèle de la composition du sang, du mécanisme de coagulation et aucune connaissance du rôle du cœur dans la circulation du sang. Bien entendu, une telle représentation profonde n'est pas toujours indispensable pour trouver le bon diagnostic, mais ceci peut être une sérieuse limitation lorsqu'il s'agit d'*expliquer* les résultats obtenus.

Une deuxième limitation provient du faible niveau d'abstraction utilisé pour décrire le processus de résolution. De fait, le système est incapable de donner une vue globale du raisonnement poursuivi et se contente ainsi de décrire le processus à travers l'enchaînement des règles, ce qui est insuffisant pour décrire correctement le comportement d'un système à base de connaissances.

Les explications fournies par les systèmes experts de première génération ne sont donc ni naturelles, ni facilement compréhensibles pour l'utilisateur.

6.3. Manque de robustesse

Les SE1G présentent également le défaut d'être fragiles. Il faut entendre par là que si on leur reconnaît une certaine compétence sur leur domaine d'expertise, ils tendent à exhiber des comportements aberrants aux frontières de ce domaine. Ceci peut être opposé au comportement des experts humains qui, lui, va en se dégradant progressivement au fur et à mesure qu'ils s'éloignent de leurs domaines d'expertise. De plus les experts seront en général capables d'apprécier leurs limites et sauront éventuellement se réfugier derrière un prudent "je ne sais pas" plutôt que de vouloir à toute force appliquer leurs savoir-faire.

Les SE1G n'ont malheureusement ni ce recul, ni cette humilité. De manière générale donc, il est reproché à ces systèmes de ne pas avoir conscience de leurs limites et de ne pas savoir changer de stratégie pour s'adapter au problème.

A fortiori, ils sont évidemment incapables de mettre en œuvre des méthodes plus générales lorsque leurs connaissances spécifiques ne sont plus pertinentes. Mais la fragilité de ces systèmes se manifeste également lorsqu'il s'agit de les faire évoluer, soit parce que le problème lui même évolue, soit, plus simplement, parce que l'on souhaite corriger un comportement non satisfaisant. La théorie voudrait qu'il suffise d'ajouter ou de modifier une connaissance. Dans la pratique, les règles se trouvent si inextricablement reliées les unes aux autres qu'il devient difficile, sinon impossible, de maîtriser leurs interactions.

L'indépendance des règles, si souvent proclamée, est en fait une illusion ; toute modification de la base de connaissances risque de provoquer des modifications inattendues du comportement, par effet de bord. En particulier, l'ajout d'une nouvelle connaissance peut très bien avoir pour conséquence que le système ne soit plus capable de résoudre des problèmes qu'il savait résoudre auparavant.

6.4. Modification difficile des stratégies de contrôle

Les stratégies des interpréteurs de connaissances sont généralement programmées (procédurales) ; or, une partie de l'expertise dans un domaine concerne aussi les stratégies de raisonnement et pas seulement les connaissances. Il serait donc souhaitable de pouvoir représenter ces stratégies par des règles de production ; cela faciliterait ainsi leur amélioration et leur apprentissage.

6.5. Réutilisabilité

Enfin, parce que les systèmes experts de première génération combinent de manière implicite les tâches à réaliser, les méthodes et les connaissances du domaine, il est impossible de réutiliser telle méthode ou tel modèle d'une application à une autre. Ceci est d'autant plus pénalisant que de tels systèmes restent, malgré tout, extrêmement difficiles à construire.

7. Conclusion

Ce chapitre nous a permis de nous familiariser avec l'Intelligence Artificielle (IA) et plus particulièrement avec les systèmes à base de connaissances (SBC). Nous avons défini ce qu'est un système expert qui se caractérise par ses objectifs et sa structure qui est composée d'une base de connaissance (BC), une base de faits (BF) et d'un moteur d'inférences (MI).

Différents formalismes de représentation de connaissances ont été cités ainsi que des exemples de systèmes experts.

Nous avons conclu à la fin que l'approche utilisée (adoptée) dans les systèmes experts de première génération (SE1G) pour le développement de SBC a conduit à de nombreuses difficultés (acquisition des connaissances, explication, robustesse, ...). Ces problèmes ont donné lieu à de nombreux travaux de recherche, et plusieurs solutions ont été proposées. Dans la suite de nos travaux nous allons nous focaliser sur le problème de l'explication qui constituera une motivation et un vecteur de progrès importants pour bon nombre de travaux sur les systèmes experts de deuxième génération (SE2G).
Nous verrons ainsi comment les explications fournies par les systèmes experts de première génération peuvent être significativement améliorées.

Chapitre II

Explication dans les systèmes à base de connaissances

1. Introduction

Depuis leur apparition, les systèmes experts ont été crédités de la propriété de « transparence », de par leur structure propre. La séparation entre la base de connaissances et le moteur d'inférences, la représentation déclarative et granulaire des connaissances et la simplicité du cycle d'inférences laissaient penser que les systèmes experts pourraient aisément expliquer leur raisonnement en termes compréhensibles par l'utilisateur.

On peut considérer que les premières recherches concernant directement les explications dans les systèmes à base de connaissances sont celles de Clancey et de Swartout. Tous les deux sont partis de systèmes experts existants auxquels ils ont ajouté des fonctions d'explication.

Les rudiments de l'explication de MYCIN furent mises en œuvre par Shortliffe[76] sous la forme de la commande RULE permettant, lors d'une session, d'afficher en langage naturel la règles courante. Les faiblesses de ces explications ont été rapidement détectées par Clancey [79]. Le travail de Clancey, dans le but d'améliorer l'explication, a d'abord consisté à réexaminer la base de connaissances de MYCIN afin d'établir le rôle de chacune des connaissances et de déterminer comment elles sont utilisées les unes par rapport aux autres. Ainsi, une nouvelle architecture du système MYCIN a été proposée : le système NEOMYCIN.

L'approche de Swartout [83] pour réaliser le système XPLAIN est complètement différente de celle utilisée par Clancey. Néanmoins, il est parti de la même critique que celle faite à l'encontre de MYCIN : incapacité du système à justifier le résultat aussi bien au niveau factuel qu'au niveau stratégique. Swartout proposa alors un générateur de système explicateur capable d'expliquer la stratégie en cours de session ou en fin de session.

Après ces précurseurs, de nombreux chercheurs ont abordé la problématique de l'explication dans les systèmes à base de règles. Deux principaux axes ont été dégagés :

- La structuration de la connaissance et l'introspection
- La modélisation du raisonnement explicatif

2. Notion d'explication

2.1. Différentes acceptions du terme « explication »

Le terme d'explication possède de nombreuses acceptions correspondant à des problématiques de recherche diverses.

2.1.1. Explication interne *versus* explication externe

Une première acception du terme concerne l'explication en tant que « ce qui rend compte d'un fait », c'est-à-dire le motif, la cause, la raison. C'est le sens donné à la démarche scientifique qui vise à expliquer le monde en tentant d'établir les causes des phénomènes observables. Dans un système expert de diagnostic comme CHECK [Console et al 88], l'objectif est d'identifier un lien causal entre des faits observables et des hypothèses de diagnostic, qui rend compte, c'est-à-dire qui explique, les symptômes. Cette acception du terme est appelée « explication interne » par [Rousset 90]. La communauté d'apprentissage

Dans le premier cas, les explications du comportement effectif de la machine sont utiles pour aider à corriger la base de connaissances, pour valider des connaissances expertes à partir de leurs justifications par rapport à la connaissance profonde ou pour aider le concepteur d'une base de connaissances à corriger les incohérences détectées.

Les explications qui s'adressent à l'utilisateur final ont pour but de lui fournir des informations sur le comportement du système, et de le convaincre de la pertinence des raisonnements suivis. Un des objectifs des systèmes à base de connaissances étant d'assister les humains dans leurs tâches complexes, et sachant que ces derniers prendront finalement la plupart des décisions, il est important que la machine puisse justifier et expliquer ses conclusions, un résultat brut n'ayant dans cette situation que peu d'intérêt.

La figure II.1 récapitule les différents sens du terme et les problématiques de recherche associées.

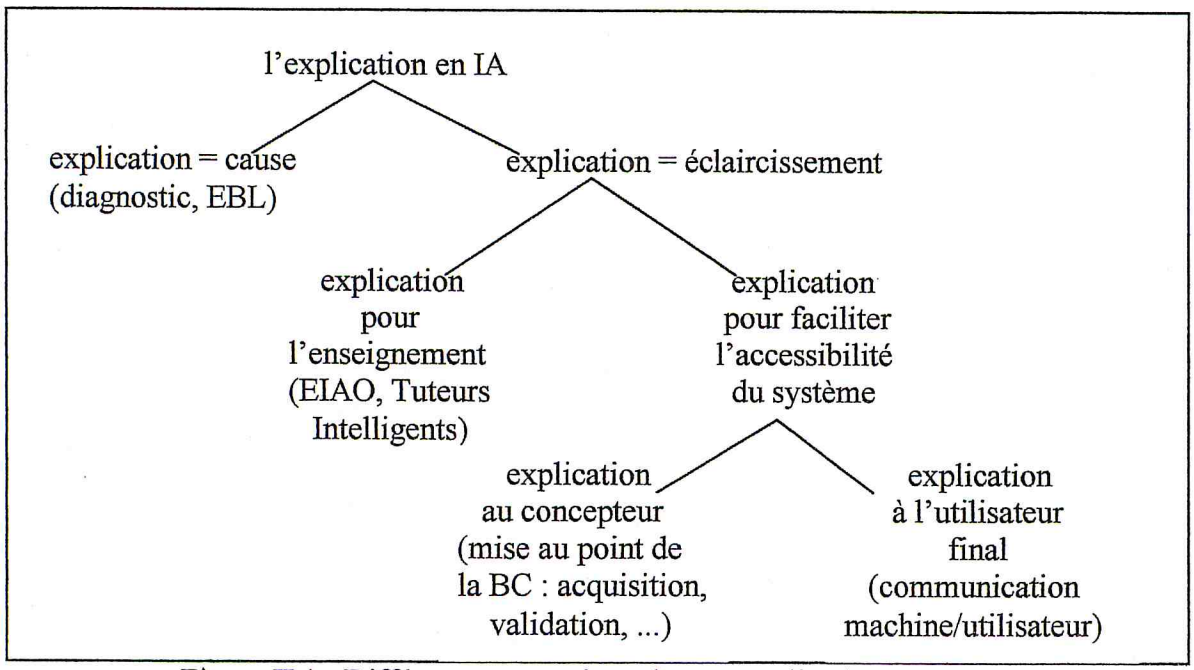


Figure II.1 - Différentes acceptions du mot explication en IA

2.2. Caractérisation de la recherche en explication

Le concept d'explication dans les systèmes à base de connaissances est né avec le premier système expert : MYCIN [Buchanan et al 84]. La représentation des connaissances sous forme de règles de production déclaratives permet, de par son caractère simple et uniforme, de rendre le système transparent à l'utilisateur.

Les premières recherches en explication sont donc issues de la communauté « systèmes experts », un peu comme un effet de bord de la représentation déclarative. Par la suite, d'autres domaines apportent leur contribution à ce nouvel édifice ; [Hasling et al 84] les classifient en deux thèmes : l'*aspect rhétorique* correspondant aux recherches en langage naturel et la *modélisation de l'utilisateur* correspondant aux recherches en sciences cognitives.

2.2.1. L'aspect rhétorique

Une fois le contenu des explications déterminé, il faut le mettre en forme pour qu'elles soient compréhensibles. On peut isoler plusieurs voies de recherches complémentaires.

Une première approche est celle des logiciels de génération de textes. Ces recherches sur le langage naturel soulignent l'importance de notions comme la structure du discours, les stratégies de communication, les préférences de l'utilisateur.

L'explication par l'exemple, l'explication par l'analogie, correspondent à autant de stratégies rhétoriques.

Le travail de Weiner en est un exemple. Son point de départ est une analyse détaillée de transcriptions de conversations, dans le but de construire un modèle des conversations humaines. Dans son système BLAH [Weiner 80], il structure les explications d'après les principaux types de justification : fournir une raison, donner des exemples, éliminer des choix alternatifs.

Signalons toutefois que les SBC fournissant une facilité d'explication n'utilisent jusqu'à présent que des générateurs de textes très simples.

Mais le langage n'est pas le seul "médium" : le graphisme permet de présenter de façon synthétique des informations. De plus en plus, des graphiques sont liés à du texte, reflétant les nouvelles possibilités graphiques des postes de travail

2.2.2. La modélisation de l'utilisateur

Dans plusieurs sous-domaines de l'IA, la question d'ajuster les réponses du programme à l'utilisateur est posée : les logiciels traitant le langage naturel, les programmes de génération de textes, et les systèmes d'EIAO (Enseignement Intelligement Assisté par Ordinateur) en sont les principaux exemples.

Les recherches en EIAO ont ainsi mis en avant l'intérêt de mémoriser les réponses de l'étudiant au cours des sessions pour mieux adapter l'aide fournie à son niveau, à ses connaissances. Il s'agit d'un objectif évident de pédagogie !

L'approche la plus courante a consisté à considérer les connaissances de l'utilisateur-étudiant comme un sous-ensemble de celles de l'expert : c'est en particulier le cas pour les systèmes d'enseignement SCHOLAR, WEST et GUIDON. Les recherches de Brown et Burton, sur le système BUGGY, ont d'autre part montré l'intérêt de représenter également les connaissances « incorrectes » de l'élève pour mieux comprendre ses erreurs.

Dans le domaine des systèmes experts, la capacité du système à s'exprimer en des termes compréhensibles en choisissant des concepts connus de l'utilisateur, est habituellement jugée comme un facteur important pour la qualité des explications.

D'autant que le SE ne raisonne pas à la fois comme un expert et comme l'utilisateur : ses connaissances sont celles de l'expert, il lui faut donc adapter ses explications.

La question posée est de savoir si un modèle détaillé de l'interlocuteur est vraiment nécessaire ?

[Swartout 85] pense que des connaissances générales suffisent, s'appuyant pour cela sur la constatation que nous n'avons généralement pas de connaissance détaillée de nos interlocuteurs.

Il semble en revanche, que la capacité de reformuler autrement l'explication, si le système à l'impression de ne pas avoir été compris, serait suffisante.

Swartout évoque donc la question des informations en retour et de la capacité du SE à dialoguer en langage naturel avec l'utilisateur.

La communauté « systèmes à base de connaissances », au sein de laquelle nous nous inscrivons clairement, possède cependant une spécifité dans son approche du problème, à savoir, *l'explication de raisonnement.* Ce type d'explications est habituellement délaissé par les autres communautés qui préfèrent porter leur effort sur la tâche d'explication plutôt que sur les problèmes complexes de représentation de raisonnements.

Nous pensons que cette différence entre *explication de raisonnements* et *explication d'éléments factuels*, tel que description ou définition, se situe essentiellement au niveau de l'accessibilité de l'objet à expliquer : les mécanismes d'explication sont généralement similaires. L'intérêt de l'explication de raisonnements réside davantage dans la difficulté d'identification et de représentation du contenu à expliquer.

3. Historique de l'explication dans les systèmes experts

3.1. Introduction

Malgré ses limites, la forme primitive de l'explication (sous forme de trace de raisonnement), proposée par les premiers systèmes experts a été considérée comme un avantage exclusif des systèmes experts par rapport aux autres technologies ; de plus, selon [Chandrasekaran et al 88] ces explications ont permis de dégager trois idées essentielles sur les types d'explications que doit pouvoir fournir un système explicatif [Richards 03] :

- l'explication de la mise en œuvre des règles, c'est-à-dire expliquer pourquoi le système a besoin de certaines données, comment une conclusion intermédiaire est obtenue ;

- l'explication des stratégies de résolution de problèmes et de leur contrôle c'est-à-dire une justification des choix d'application des règles.

- l'explication des connaissances utilisées, c'est-à-dire une justification de la base de connaissances du système ;

Le premier type est apparu avec MYCIN [Shortliffe 74], le deuxième a été développé dans NEOMYCIN [Clancey et al 81] et le troisième type a été concrétisé dans le système XPLAIN [Swartout 83].

3.2. MYCIN

[Saïdi 92] [Kassel 86]

La plupart des travaux sur l'explication font référence aux capacités d'explication du système MYCIN ; nous débutons donc notre étude par ce pionnier des systèmes experts.

3.2.1. Une nouvelle architecture de logiciel

La caractéristique essentielle de MYCIN consiste en la séparation entre les connaissances spécifiques de la tâche et les mécanismes chargés d'exploiter ces connaissances. Réunis au sein du moteur d'inférences, les mécanismes d'exploitation sont indépendants du domaine d'application.

Rappelons que cette séparation était une réponse au problème de la gestion, par un programme informatique, de grandes quantités de connaissances : elle introduit un style de programmation radicalement nouveau.

Le choix des règles de production, comme formalisme de représentation des connaissances, était avant tout (1972) le choix du raisonnement symbolique : il fut guidé par le souci de faciliter la gestion (acquisition puis évolution) de la base de connaissances.

3.2.2. Le concept de transparence

Mais l'idée clé de MYCIN est celle de transparence : rendre le système compréhensible et ce, malgré ou à cause de la complexité de la tâche qu'il accomplit et du grand nombre de connaissances qui fondent ses décisions. D'où le qualificatif de « boîte de verre » qui sera attaché aux systèmes experts.

Dans la thèse d'Edward Shortliffe, cette notion restera essentiellement passive : l'utilisateur pouvait à tout moment connaître la règle invoquée, par la commande RULE, le logiciel se bornant à imprimer cette règle. Cependant, au cours des années 1973-76, l'idée de transparence va évoluer et faire appel à la capacité du système à comprendre ses propres opérations [Buchanan et al 84] : comme conséquence de la nouvelle architecture, le système possède dans une certaine mesure des connaissances sur ses connaissances et la façon dont il les interprète. Cette capacité s'exprimera dans deux modules adjoints au système expert : les modules d'explication et d'acquisition des connaissances.

En fait, ces deux facilités greffées au système ne servent qu'un seul but : permettre à l'expert de faire évoluer la base de connaissances. C'est d'ailleurs Randall Davis qui développera les capacités d'explication de MYCIN en même temps qu'il travaillera sur TEIRESIAS, le module d'acquisition des connaissances. Les explications sont une aide offerte à l'expert pour que celui-ci se rende compte de ce que MYCIN sait ou ne sait pas, et de la façon dont MYCIN utilise ses connaissances.

Ce n'est que plus tard que les facilités d'explication seront jugées également importantes pour l'acceptation du logiciel auprès de l'utilisateur final.

3.2.3. Les explications

Si le but initial de MYCIN n'a pas été d'expliquer mais d'aider à faire du diagnostic, très vite la nécessité d'introduire des capacités explicatives s'est fait sentir. En effet Shortliffe mentionne l'existence, dans MYCIN, d'une première capacité d'explication rudimentaire qu'il a créée : la commande RULE. Cette commande qui ne peut être utile que pour les développeurs du système, permet tout simplement d'afficher la règle courante en Lisp.

C'est en travaillant sur le système d'acquisition des connaissances TEIRESIAS associé à MYCIN que Davis proposa d'autres commandes afin d'offrir des capacités explicatives aux utilisateurs du système. Il mit en œuvre les commandes WHY et HOW. Avec ces commandes, l'utilisateur peut poser des questions sur le contenu de la base de connaissances ainsi que sur les décisions prises par le système, c'est-à-dire la façon dont les faits ont été déduits.

Nous donnons ci-dessous quelques exemples de questions avec les réponses fournies par MYCIN. Ces exemples sont tirés de [Farreny 85] : le texte de la question tapé au clavier par l'utilisateur, est reproduit en caractères gras.

PRESCRIVEZ-VOUS PARFOIS DE LA CARBENICILLINE POUR LES INFECTIONS PAR PSEUDOMONAS ?

REGLES CONCERNEES : REGLE024 REGLE068
REGLE137 REGLE138

LESQUELLES DESIREZ VOUS VOIR ?

REGLE024

<suit le texte de la règle>

Figure II.2 - Une question sur le contenu de la base de connaissances

En cours de session, l'utilisateur peut demander au système pourquoi celui-ci requiert de l'information (figure II.3) :

POURQUOI

JE CHERCHE A DETERMINER L'IDENTITE D'ORGANISME-1
JE SAIS DEJA QUE :

LE SEUIL D'ENTREE DE L'ORGANISME EST LE GU SUPERIEUR
LA MORPHOLOGIE DE L'ORGANISME EST DU TYPE BATONNET
LA COLORATION DE L'ORGANISME EST GRAM-
LE SITE DE LA CULTURE EST LE SANG

PAR CONSEQUENT,

SI : 1) LE PATIENT N'A PAS SUBI DE MANIPULATION OU D'OPERATION GENITO-URINAIRE,
2) LE PATIENT N'A PAS ETE TRAITE POUR UNE INFECTION DE L'APPAREIL URINAIRE

ALORS : IL EXISTE UNE EVIDENCE SUGGESTIVE (0.6) QUE L'IDENTITE DE L'ORGANISME SOIT E.COLI

Figure II.3 - Une explication du type POURQUOI (WHY)

En fin de session, l'utilisateur peut demander à MYCIN comment un fait a été déduit (figure II.4) :

COMMENT AVEZ-VOUS DECIDE QU'ORGANISME-1 POUVAIT ETRE UN PSEUDOMONAS ?

J'AI UTILISE REGLE085 POUR CONCLURE QUE L'IDENTITE D'ORGANISME-1 EST PSEUDOMONAS. CECI A DONNE UN FACTEUR DE CERTITUDE CUMULE DE 0.6

Figure II.4 - Une explication du type COMMENT (HOW)

Pour répondre à cette dernière question, le module d'explication utilise un historique de la consultation : la mémorisation des faits déduits avec l'origine de leur déduction.

Les questions sont interprétées par le système en termes de règles et de faits : MYCIN présente alors la règle qui aura utilisé, ou permis de déduire, le fait en question.

3.2.4. MYCIN : un Système Expert Explicatif

Les explications, nous l'avons rappelé, ne faisaient pas partie des préoccupations principales des concepteurs de MYCIN. Pour traduire la façon dont cette capacité a été introduite : un module d'explication a été adjoint au système expert, nous parlerons de Système Expert Explicatif (noté par la suite SEE).

Quant à la méthode d'explication, elle consiste nous l'avons vu, à présenter la règle utilisée pour déduire un fait. Nous qualifierons cette explication d' « **explication trace** » ou de simple présentation du raisonnement.

En résumé :

La capacité d'explication de MYCIN tient essentiellement à la nouvelle architecture du logiciel. En ce qui concerne la méthode d'explication, MYCIN se contente de ne présenter qu'une seule étape de son raisonnement en exhibant la règle de production correspondante. C'est une simple présentation de la trace de raisonnement.

3.3. NEOMYCIN

[Saïdi 92] [Kassel 86]

Le point de départ de cette recherche est la transformation de MYCIN en un système d'enseignement : GUIDON [Clancey 82].

Les performances de MYCIN, tant dans sa tâche de diagnostic que dans sa capacité à fournir des explications de son raisonnement ont suggéré que sa base de connaissances puisse servir de matériel de départ pour l'enseigner à des étudiants de médecine.

Ce projet, consistant à réutiliser l'expertise à des fins d'enseignement, a mis en évidence des limitations de MYCIN, notamment de ses capacités explicatives, et donné naissance au système NEOMYCIN (NEO pour Nouveau).

Clancey dégagea trois faiblesses de MYCIN :

- pas de justification des connaissances utilisées,
- la stratégie de résolution n'est pas explicitée,
- mauvaise représentation des structures.

et s'efforça d'y remédier.

3.3.1. Justification des connaissances

Le terme « justifier » est à prendre ici au sens : expliquer ce sur quoi repose la connaissance.

Ce type d'explication doit permettre à l'utilisateur d'avoir une autre vue sur la connaissance, de se convaincre que celle-ci est correcte. L'objectif visé par Clancey était d'ordre pédagogique : le fait de fournir la raison d'une connaissance à un étudiant doit lui permettre de mémoriser plus facilement cette connaissance.

Mais la raison ultime de cette justification vient de la constatation suivante : l'expert qui a communiqué ses connaissances au système et l'utilisateur final ne raisonnement pas nécessairement de la même façon, n'utilisent pas les mêmes connaissances. Et cette différence ne se réduit pas à un problème de vocabulaire : en particulier, l'expérience de MYCIN montre que certaines règles fournies par l'expert nécessitent de la part de l'étudiant plusieurs pas d'inférences. On peut parler dans ce cas de connaissances « compilées » dans la mesure où ces règles constituent pour l'étudiant des raccourcis de raisonnement.

Ceci a conduit Clancey à un ré-examen approfondi de la nature et du contenu des règles de MYCIN. Il a ainsi jugé nécessaire de définir et d'expliciter les différentes hiérarchies et relations existantes entre les règles. Pour cela il a proposé de classer les règles de MYCIN en quatre catégories suivant la nature du lien existant entre leurs prémisses et leurs conclusions : les *règles d'identification*, le *règles du domaine*, les *règles du bon sens* et les *règles causales*.

- *Les règles d'identification* : ce sont les règles utilisées pour classer ou identifier des objets d'après leurs propriétés. Par exemple :

  Si la coloration de l'organisme est gram-
  et la morphologie de l'organisme est du type bâtonnet

  Alors l'organisme est peut-être de type bactérie

- *Les règles du domaine* : ces règles définissent les faits particuliers du domaine. Par exemple :

  Si un médicament a été administré par voie orale
  et s'il a été faiblement absorbé

  Alors ce médicament a été mal administré

- *Les règles de bon sens* : ces règles permettent de représenter les connaissances communément admises. Par exemple :

  Si le patient est de sexe masculin

  Alors il n'est pas enceinte

- *Les règles causales* : ce sont les règles dont les parties prémisse et action sont reliées par une relation causale. Par exemple :

  Si le patient a moins de huit ans

  Alors ne pas prescrire de tétracycline

Clancey considère que pour les trois premières catégories de règles, la nature du lien *prémisses-conclusion* fournit une explication suffisante. En revanche, expliquer le raisonnement sous-jacent d'une règle causale n'est pas aussi direct. En effet la plupart des règles causales de MYCIN représentent des complications médicales qui ne peuvent être facilement exprimées comme des relations anatomiques et des processus physiologiques. Pour

comprendre ces règles, il faut comprendre le raisonnement sous-jacent au lien causal. Ce raisonnement que Clancey appelle *connaissance de support*, n'est pas explicité dans les règles et par conséquent, le système ne peut les justifier. La dernière règle présentée contient un mécanisme sous-jacent que MYCIN n'est pas en mesure de présenter, comme le montre la figure II.5.

Emploi de la tétracycline chez l'enfant
=> nocivité du médicament sur le développement des os
=> coloration noire des dents
=> effets indésirables sur le corps

Figure II.5 - Raisonnement sous-jacent au lien causal d'une règle

3.3.2. Explication de la stratégie de résolution

Un autre aspect très important de la problématique des connaissances implicites concerne la stratégie de résolution. Dans la plupart des systèmes à base de connaissances, les règles sont structurées pour que leur enchaînement logique conduise à la solution du problème. Cependant, les connaissances sur la façon de conduire le raisonnement ne sont pas représentées explicitement [Jiménez 90]. De ce fait, le SE ne peut répondre à un certain nombre de questions sur son comportement. Par exemple : pourquoi pose-t'il telle question avant telle autre ? Y-a-t'il une raison propre au domaine ou bien cet ordre n'est-il que le reflet de l'ordre d'écriture des règles, auquel cas on peut qualifier le comportement d'arbitraire ?
La réponse à cette question est importante pour l'acquisition des connaissances, si l'on souhaite modifier le comportement du SE.

Expliquer la stratégie est pour Clancey un objectif d'enseignement il est clair selon lui qu'enseigner comment faire quelque chose diffère de simplement montrer comment ce quelque chose a été réalisé. L'étudiant qui aura par lui-même à établir des diagnostics a besoin de comprendre la stratégie : la démarche de diagnostic du SE.

Dans NEOMYCIN, Clancey a été amené à séparer la connaissance médicale de la connaissance de diagnostic. L'étape préliminaire, pour réaliser cet objectif, a consisté à cerner *le savoir stratégique* dans MYCIN. Clancey a structuré celui-ci en termes de tâches, qui correspondent aux buts et aux sous–buts de la résolution du problème et de méta–règles, qui sont les méthodes pour atteindre ces buts.

L'exemple suivant montre une méta-règle dans NEOMYCIN :

METAREGLE 037
Si il y a une donnée qui peut être demandée
et qui soit une caractéristique de l'élément considéré
Alors s'enquérir de cette donnée

3.3.3. Définition d'une meilleure représentation des structures utilisées

L'étude approfondie des règles de MYCIN entreprise par Clancey a conduit à la constatation suivante : le formalisme de représentation de la connaissance de MYCIN sous forme de règles de production est trop faiblement structuré pour bien expliquer.
Clancey a associé un type d'utilisation à chaque connaissance qui exprime dans quelles circonstances ou pour quel objectif cette connaissance est utile pour l'explication.

Trois types de connaissances ont été définis : *les connaissances structurelles*, *les connaissances de stratégie* et *les connaissances de support*.

i) *les connaissances structurelles* correspondent aux différentes hiérarchies et relations existantes entre les règles. Elles classifient la connaissance du domaine dans plusieurs groupes et explicitent toutes les relations existantes entre ces différents groupes et entre les éléments d'un même groupe.

ii) *les connaissances stratégiques* correspondent à l'ensemble des connaissances de contrôle de résolution. Les connaissances structurelles étant indexées de deux façons, la stratégie peut être soit d'évoquer certains diagnostics et de considérer les règles qui appuient ces hypothèses, soit , au contraire, d'évoquer la recherche de certaines données qui permettent d'établir des conclusions. Ce savoir stratégique apparaît dans les règles de MYCIN sous différentes formes : tâches si la stratégie concerne une classe de problèmes ; méta–règles si la stratégie est liée à un domaine particulier.

iii) *les connaissances de support* correspondent aux raisonnements sous-jacents qui relient une prémisse d'une règle à sa partie conclusion. Elles permettent de justifier les règles. Ce savoir est très précieux si l'on veut modifier la base de connaissances ou si on veut l'enseigner.

Chaque type de connaissances est stocké dans le système par l'intermédiaire d'un mode de représentation spécifique ce qui implique l'utilisation de mécanismes de contrôle plus fins et donc une trace de raisonnement plus structurée. Clancey a généralisé cet ensemble d'idées dans le système HERACLES. Il a ainsi créé un réseau de classification pour la taxonomie de maladies, implanté les règles causales sous forme de règles heuristiques et les stratégies de diagnostic médical sous forme de stratégies de contrôle.

La figure suivante représente l'architecture du système NEOMYCIN :

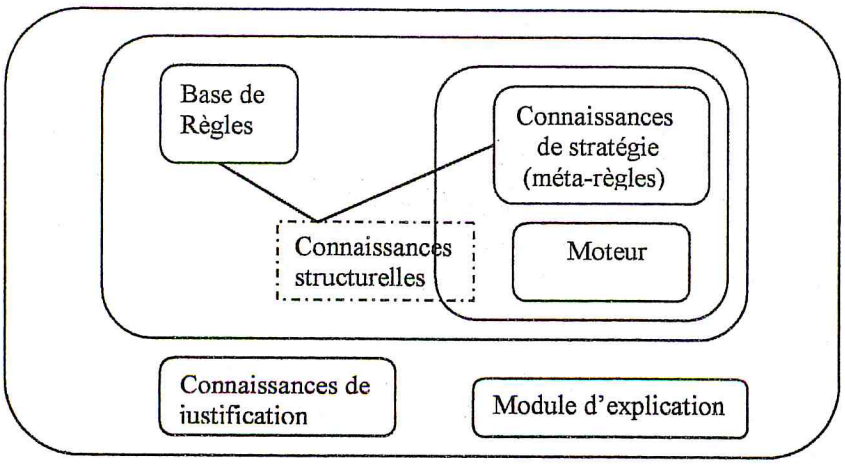


Figure II.6 - Architecture du système NEOMYCIN

Le nouveau module d'explication se contente, comme celui de MYCIN, de présenter une trace du raisonnement [Hasling et al 84]. Cependant, cette trace est différente de celle de MYCIN, puisqu'elle montre comment les métarègles se sont enchaînées pour finalement invoquer les règles : une partie de la trace correspond aux connaissances de stratégie utilisée.

En fait, chaque règle est invoquée par une méta-règle ; l'objectif de NEOMYCIN étant d'expliciter la stratégie de diagnostic du SE, à la question « POURQUOI », le module d'explication présente la méta-règle courante (ceci en cours de session). Sur l'exemple suivant (figure II.7), la première phrase correspond à une question du SE :

EST-CE QUE ALI A DE LA FIEVRE ?

POURQUOI

NOUS ESSAYONS DE DECIDER SI ALI
A UNE INFECTION
LA FIEVRE EST FORTEMENT ASSOCIEE A L'INFECTION

Figure II.7 - NEOMYCIN : explication de type POURQUOI

En résumé :

Les capacités d'explication sont placées dans NEOMYCIN au premier rang des préoccupations : elles s'adressent à un utilisateur « novice » qui n'a pas participé à la conception du SE.

Pour l'explication de la stratégie, la structure de MYCIN ne convient plus. L'idée importante ici est que n'importe quel Système Expert ne permet pas de construire un Système Expert Explicatif. Pour améliorer la capacité d'explication, on a modifié le SE mais conservé la même méthode d'explication, l'explication-trace. NEOMYCIN constitue un modèle psychologiquement plus valide qui simule de plus près la démarche de l'expert.

En particulier, un autre point important est mis en évidence : le fait que la connaissance présente dans les SE se trouve souvent sous une forme « compilée ». Ce terme illustre nous l'avons vu, la nature des connaissances de l'expert. Il caractérise aussi la façon dont les connaissances sont représentées : l'uniformité du formalisme de représentation des connaissances impose que plusieurs types de connaissances soient représentés sous un même format. Ce qui pouvait alors sembler un avantage pour la phase d'acquisition des connaissances constitue un handicap pour les types d'explication que nous venons de considérer.

Le tableau suivant (figure II.8) décrit la synthèse des évolutions MYCIN/NEOMYCIN :

MYCIN	NEOMYCIN
Raisonnement de base But → règle → sous-but	*Raisonnement de base* Tâche → méta-règle → sous-tâche
Un but est examiné pour satisfaire la prémisse d'une règle (chaînage arrière)	*Une tâche est examinée* En tant qu'action d'une méta-règle (chainage avant sur ensemble des règles)
Pourquoi un but est examiné ? règle qui l'utilise comme sous-but	*Pourquoi une tâche est appelée ?* la méta-règle qui l'invoque
Comment un but est déterminé ? les règles qui concluent sur lui	*Comment une tâche a été accomplie ?* la méta-règle qui l'a réalisée

Figure II.8 - Tableau récapitulatif de l'évolution MYCIN/NEOMYCIN

3.4. XPLAIN
[Saïdi 92]

3.4.1. Introduction :

Le point de départ des travaux de [Swartout 83] sur les explications est le système expert Digitalis Therapy/Advisor de consultations médicales (i.e. utilisation de la digitaline pour stabiliser le rythme cardiaque, produit dont les nombreux effets secondaires rendent la prescription délicate). La motivation du développement de fonctions explicatives provient de la nécessité de faire accepter les conclusions du système aux praticiens.

La principale difficulté provient de ce que les informations nécessaires pour justifier la résolution d'un problème ne sont pas incorporées au système, car ce dernier n'en a pas besoin pour résoudre le problème. Ces connaissances, encapsulées de façon procédurale, ne peuvent être utilisées par un système d'explication. Pour être certain de capturer les connaissances nécessaires pour l'explication, Swartout utilise la programmation automatique. Pour cela, il distingue deux types de connaissances :

i) les connaissances du domaine d'application,

ii) les connaissances permettant d'engendrer un programme exécutable à partir des connaissances de n'importe quel domaine d'application.

XPLAIN est donc un outil pour créer des systèmes experts et pour justifier leurs actions. Avant de présenter la structure du système XPLAIN, nous allons d'abord voir comment Swartout intègre ces deux types de connaissances dans son système, comment celles-ci vont être utilisées par le *rédacteur* pour produire le programme exécutable et enfin comment Swartout tire profit de la programmation automatique pour fournir les explications.

3.4.2. Les connaissances du domaine :

Les connaissances du domaine d'application sont composées d'un modèle du domaine et d'un modèle de principes. Le modèle du domaine contient les relations causales et les hiérarchies existantes dans l'expertise du domaine. C'est ce type de connaissance qui n'apparaît pas habituellement et qui pourtant permet de justifier les actions du système.

i) Le modèle du domaine

Ce sont des connaissances de base du domaine médical qu'apprennent les étudiants en médecine durant les premières années d'études. Elles sont modélisées par un réseau sémantique dont une partie simplifiée est présentée dans la figure II.9. Ces connaissances sont descriptives car elles contiennent les faits du domaine et les relations entre ces faits, mais elles ne fournissent aucune indication sur la manière dont elles s'utilisent.

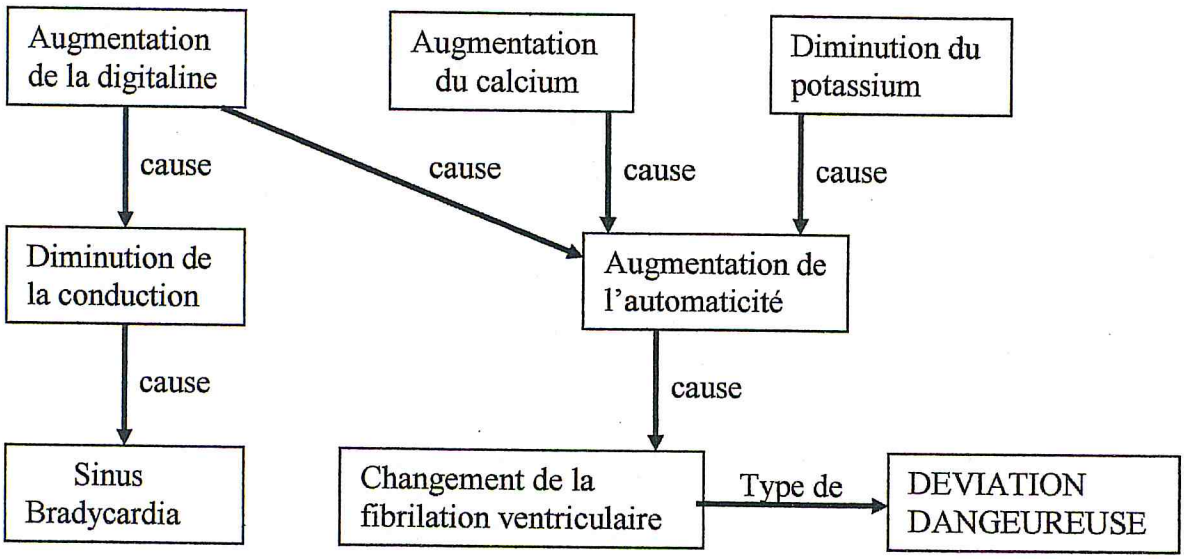


Figure II.9 - Partie simplifiée du modèle du domaine

ii) Le modèle du principe

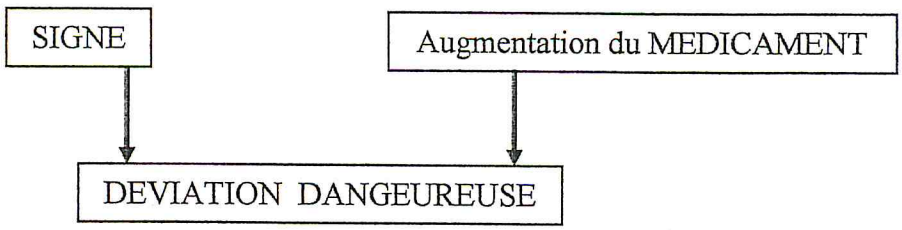
Les principes du domaine contiennent les méthodes et les heuristiques du domaine de l'expertise ; ils indiquent au *rédacteur* la méthode à suivre pour atteindre un but, par exemple « administrer la digitaline ». Ce sont des méthodes abstraites [abstract procedure schema] instanciées sur les faits du domaine pour créer des procédures spécifiques.
Chaque principe du domaine est composé d'un *but*, d'une *structure* et d'une *méthode prototype*. Le *but* indique au *rédacteur* l'usage qu'il peut faire de ce principe ; la *structure* contient les informations qui permettent d'établir un lien avec le modèle du domaine et la *méthode prototype* est une méthode abstraite indiquant comment accomplir le « but ».

La figure II.10 montre un principe du domaine.

PRINCIPE

BUT Prévoir la toxicité d'un MEDICAMENT

STRUCTURE



METHODE-PROTOTYPE

Si un SIGNE existe
Alors réduire la dose du MEDICAMENT
Sinon maintenir la dose du MEDICAMENT

Figure II.10 - Exemple de principe du domaine

Notons que ce principe contient deux variables : MEDICAMENT et SIGNE (ici en majuscules) ; il est sélectionné par le générateur de programmes si son but peut ^tre mis en correspondance avec celui de l'étape courante du processus d'affinement. Par exemple, si le but courant est de prévoir la toxicité de la digitaline, le principe ci-dessus est sélectionné en substituant digitaline à MEDICAMENT.

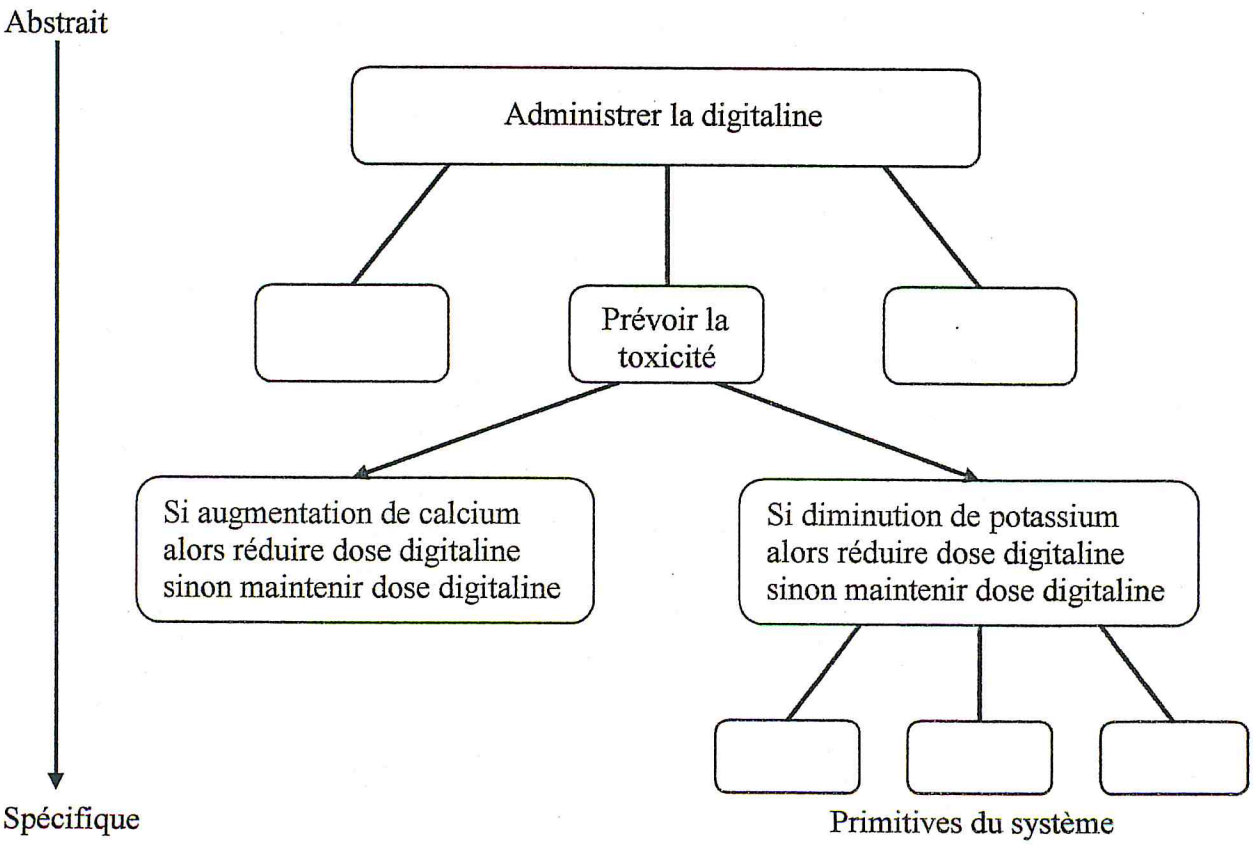


Figure II.11 - Une partie de la structure d'affinement

Le générateur automatique essaie de mettre en correspondance la structure du principe avec le modèle du domaine pour chercher toutes les instances possibles entre ces deux entités. Dans ce cas, il y a deux substitutions de la variable SIGNE : *augmentation de calcium* et *diminution de potassium*. Ces substitutions sont alors propagées sur la méthode-prototype créant ainsi deux méthodes. Dans la première, SIGNE est substitué par « diminution de potassium » et dans la seconde, SIGNE est substitué par « diminution de potassium ». Ces méthodes sont ensuite placées dans la structure d'affinement comme le montre la figure II.11.

3.4.3. Le rédacteur :

Le rédacteur engendre des sous-buts de plus en plus spécifiques, jusqu'à ce qu'il atteigne le niveau des primitives qui constituent les feuilles de l'arborescence (ces primitives n'ont généralement pas besoin d'être expliquées). Un interpréteur exécute ensuite le programme construit, en laissant une trace dans une pile de contrôle, pour l'explication.

La figure II.12 montre le schéma fonctionnel du système XPLAIN avec, entre autres, les modules « générateur automatique de programmes » et « générateur de langage naturel ».

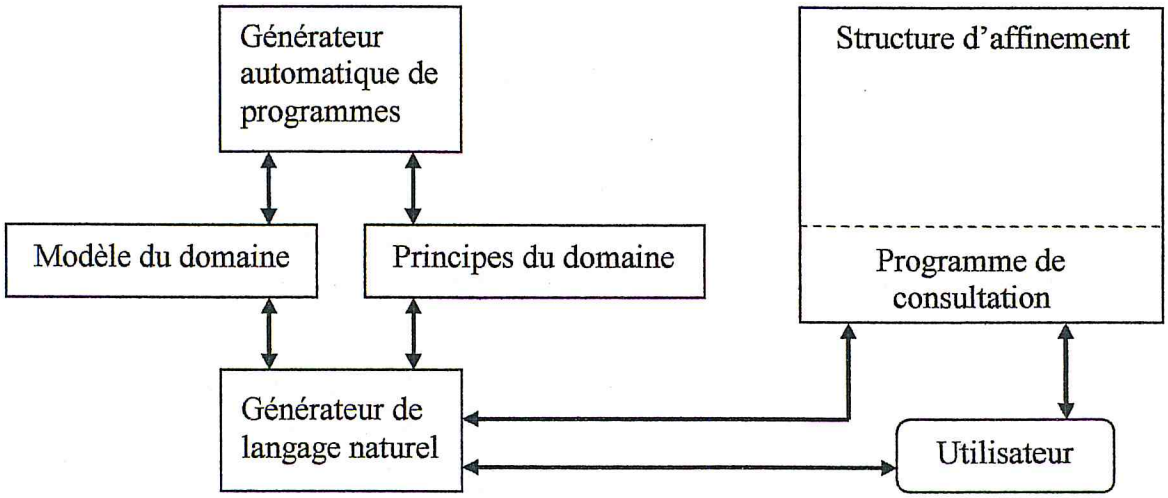


Figure II.12 - Vue d'ensemble d'XPLAIN

3.4.4. Les explications d'XPLAIN :

Le premier mécanisme lié au processus de génération d'explications est celui des points de vue : à chaque nœud de la structure d'affinement est associé un point de vue (Médecin, Table, Ordinateur...). Avant chaque explication, deux listes de points de vue - à inclure et à exclure – sont données. Ainsi, l'adéquation des explications à la requête de l'utilisateur est-elle réalisée en incluant ou excluant les étapes du raisonnement concerné, suivant le point de vue associé.

XPLAIN peut décrire la stratégie qu'il est en train de poursuivre et la justifier, il peut aussi donner une vue d'ensemble de cette stratégie ou encore décrire les méthodes de résolution utilisées à différents niveaux d'abstraction. En réponse à une question

POURQUOI, il remonte de l'étape courante à la première procédure dite « de plus haut niveau » dont le point de vue convient (ou qui n'a pas de point de vue spécifié), en sautant les procédures utilisant les primitives systèmes. Le générateur crée alors l'expression associée à la procédure de plus haut niveau. Si celle-ci a été affinée en utilisant une *raison d'être*, la raison d'être est exprimée. L'exemple II.13 montre une trace d'explication fournie par XPLAIN.

Comme dans NEOMYCIN, l'étude approfondie des différents types de connaissances utilisées et l'explication des rôles de chacune d'elles permettant de fournir de bonnes explications. XPLAIN au même titre que NEOMYCIN peut commencer par décrire sa démarche à un haut niveau d'abstraction en termes de buts et de méthodes générales en évitant de se répéter et de faire des analogies entre des justifications comme le souligne [Safar 87].

Comment as-tu réduit la dose pour l'augmentation sérum calcium ?

La dose après ajustement de l'augmentation du sérum calcium a été calculée suivant la formule suivante :
D2 = D1 C
Où :
C = la constante de réduction de l'augmentation du sérum calcium (0.8)
D1 = la dose avant ajustement de l'augmentation du sérum calcium (0.25)
D2 = la dose après ajustement de l'augmentation du sérum calcium (0.20)
La dose de digitaline ajustée pour la condition du muscle du cœur, sérum potassium et sérum calcium a été fixée à 0.20.

Figure II.13 - Visualisation de la trace d'XPLAIN

5. Principaux axes de recherche

5.1. Pourquoi dissocier l'explication du système à base de connaissances ? [Lemaire 92]

La prise en compte de l'explication de raisonnement conduit donc à distinguer deux entités correspondant aux deux termes de cette expression, à savoir, un module effectuant un raisonnement (appelé système expert ou résolveur de problèmes) et un module chargé de l'explication. Cette distinction explicite n'est pas triviale et il est important d'en préciser les raisons et les limites.

Historiquement, l'explication est tout d'abord apparue avec MYCIN comme un effet de bord de la représentation déclarative des raisonnements ; [Kassel 86] parle de conséquence « heureuse » de cette représentation.

La préoccupation première de ces pionniers n'était donc pas l'explication et ceci justifie la séparation entre explication et raisonnement qui en résulte. D'autre part, les

mécanismes d'explication imaginés se sont révélés complètement génériques : ils dépendaient du formalisme de représentation des raisonnements mais pas du tout des connaissances du domaine. Le module d'explication pouvait donc être adjoint sans problème à un autre système expert du même type.

Cette première justification, d'ordre historique, se complète par un argument de sens commun. Le raisonnement et l'explication font référence à des connaissances bien distinctes qui ont une sémantique claire (cela facilite d'autant la distinction). [Collins 92] fait d'ailleurs référence à un travail de Evans et Wason qui prouve que les gens montrent une propension remarquable à produire des justifications d'une solution, même quand celle-ci est fausse. Ils suggèrent que la rationalisation alléguée pour justifier une décision ne correspond pas nécessairement au processus intellectuel qui y a mené.

L'argument le plus significatif est cependant la nécessité de décomposer en deux unités distinctes un problème bien souvent complexe, à savoir, la communication entre un système dit « intelligent » et un humain. Suivant la stratégie bien connue, « diviser pour régner », ce comportement est alors scindé en un module dédié au raisonnement et un à l'explication, ce qui facilite l'étude de chaque entité prise séparément. Cette stratégie est cependant intéressante dés lors que les éléments décomposés se révèlent faiblement couplés. Or, certaines situations d'interaction avec l'utilisateur combinent raisonnement et explication et peuvent remettre en cause cette distinction. Supposons ainsi que le résolveur de problèmes ait abouti à une conclusion par des méthodes heuristiques. L'utilisateur demande des explications sur ce résultat. En fonction du contexte, le système peut juger opportun de justifier ces raccourcis heuristiques par un chemin causal qu'il obtiendra grâce à un nouveau raisonnement, cette fois-ci profond. C'est donc une situation dans laquelle on effectue un raisonnement uniquement pour des besoins d'explication. La dichotomie entre module d'explication et module de raisonnement n'est donc pas toujours nette, ces deux entités pouvant être fortement dépendantes l'une de l'autre. Il n'apparaît donc pas trivial de les étudier chacune de manière isolée.

La conclusion de cette partie est que la séparation entre raisonnement et explication, certes bien acquise dans la communauté IA, relève davantage d'une hypothèse de travail

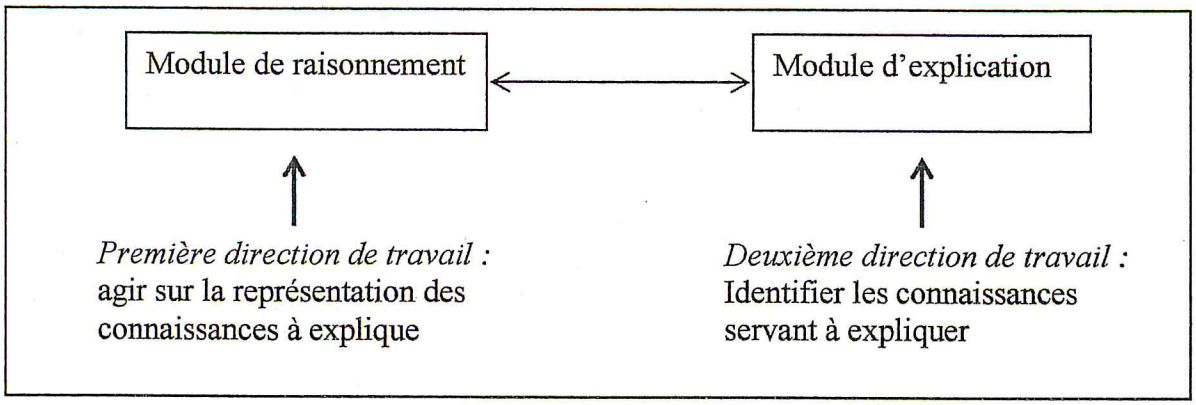


Figure II.14 - Deux directions de recherche en explication

simplificatrice plutôt que d'une conclusion définitive. Elle pourra et devra être transgressée si la situation l'exige. On pourrait même envisager, lorsque les recherches en explication auront progressé, d'étudier le système explication-raisonnement comme un tout, et de nouvelles caractéristiques naîtront sûrement de cette vision globale. Mais nous n'en sommes pas encore là, la distinction existe et nous l'utilisons comme telle. Cette séparation entre explication et raisonnement donne alors naissance à deux courants de recherche en explication, selon que l'on travaille sur l'une ou l'autre de ces entités (figure II.14).

5.2. Première direction : structuration et introspection [Saïdi 92] [Lemaire 92]

Le premier axe de travail consiste à travailler sur le module de raisonnement et notamment sur la représentation de ses connaissances. L'idée de base de cette approche est la suivante : plus les connaissances à expliquer sont explicitées et représentées clairement, plus la tâche du module d'explication en est facilitée. On retrouve là le paradigme d'explication fondé sur le concept de transparence : explicable est alors synonyme d'explicite. Explicite signifie ici accessible au module d'explication. Plus les connaissances sont représentées explicitement, plus les mécanismes d'analyse de la trace de raisonnement (l'introspection selon [Chandrasekaran et al. 88]) en sont simplifiés. Les règles de production se sont ainsi révélées une représentation explicite pour le module d'explication, en raison de leur simplicité et de leur uniformité.

L'explicitation des connaissances pour l'introspection n'est pas la seule façon d'améliorer les capacités d'explication. Un autre axe de travail consiste à décrire les connaissances à expliquer en des termes directement compréhensibles par l'utilisateur, et ayant un sens pour lui. C'est toujours de l'explication, mais cette fois-ci en direction de l'utilisateur. Cette approche fait référence à ce que [Newell 81] appelle le « niveau connaissance » (*knowledge level*). Il s'agit de décrire le raisonnement à un niveau plus abstrait, indépendant des formalismes de représentation informatique. L'objectif est de pallier les limitations de l'explication en termes de règles, qui n'apparaît pas naturelle pour un humain, et d'expliquer en termes de concepts de raisonnements de plus haut niveau. On retrouve là tout le courant des systèmes experts dits « de deuxième génération ».

Le système NEOMYCIN illustre parfaitement cette direction de travail. MYCIN n'était pas adapté pour être utilisé en tant que système d'enseignement car son raisonnement en termes de règles conduisait à des explications peu naturelles. En décrivant ce raisonnement en termes de tâches de haut niveau, Clancey a obtenu des explications de meilleure qualité, sans avoir modifié notablement les mécanismes d'explication.

5.2.1. C.Q.F.E (Ce Qu'il Fallait Expliquer)

Pour Kassel [86], la plupart des systèmes experts classiques se contentent de présenter pas à pas les inférences qu'ils ont réalisées pour arriver à la solution. Ce processus est fastidieux et peu efficace pour l'utilisateur, car il ne lui permet pas d'avoir une vue d'ensemble sur le raisonnement.

L'auteur propose de modifier la trace, d'une part, parce qu'elle est loin de toute structure d'explication humaine et d'autre part, parce que toutes les inférences ne présentent pas forcément le même intérêt. Ces deux idées sont à la base de la réalisation du système C.Q.F.E.

5.2.1.1. La connaissance manipulée

C.Q.F.E. exploite deux types de connaissances :

i) *une connaissance de surface* qui correspond à l'expertise en résolution de problèmes. Cette connaissance est représentée sous forme de règles de production. Elle correspond au savoir-faire de l'expert pour résoudre un problème. Par exemple :

Si (interrupteur-ma ?x ?y)
(sur ?y ON)
(voyant-alim ?x ?z)
(etat ?z eteint)
Alors (non (alimenté-en-courant ?x))

ii) *une connaissance profonde* qui décrit les objets et les concepts qui organisent ce savoir-faire de l'expert. Cette connaissance est représentée dans un formalisme objet et constitue le modèle conceptuel. Elle n'est utilisée qu'en fin de session par le module d'explication.

Chaque concept est défini par un certain nombre d'attributs qui le caractérisent. Il est possible de définir des sous-types d'un concept, aux quels sont associée des caractéristiques propres. Les objets du domaine d'application correspondent à des instances particulières des concepts nommées *représentants.*

Voici la syntaxe utilisée pour définir le type ' appareil électrique' et le *représentant* « Versatec » :

```
(deftype appareil-electrique ~ appareil
        (sous-tension ~ slot
                (domaine = '$booléen))
        (voyant –alim ~ slot
                (domaine = 'voyant)
                (mode= 'uni)))
```

La versatec est une imprimante à aiguilles reliée à un boîtier multiprise. Ce représentant hérite de plus les propriétés d'un appareil électrique, c'est-à-dire qu'il comporte un voyant d'alimentation et qu'il peut être sous tension.

```
(defrep versatec ~  imprimante
        (type = 'à-aiguilles)
        (relié-à = 'boîtier-multiprise))
```

D'autres types de connaissances décrivant les états possibles des objets du domaine sont aussi représentés explicitement.

Ainsi, pour décrire qu'un *interrupteur marche-arrêt a deux positions qui sont ON et OFF et que dans les conditions de bon fonctionnement, il est sur ON,* on décrit la structure suivante :

```
(defent interrupteur ~ élément
        (sur ~ slot
            (valeur = '(ON OFF)
            (fonct-normal = ' ON))
```

Ces connaissances sont utilisées pour repérer, parmi les objets manipulés durant le raisonnement, ceux dont le fonctionnement est anormal ou atypique.

5.2.1.2. Le module d'explication

Ce module permet de répondre à l'utilisateur qui s'interroge, après la session, sur les raisons de la déduction d'un fait donné. La production d'explication par C.Q.F.E se compose de trois étapes :

- structuration de la trace en blocs de raisonnement,
- recherche des faits pertinents,
- rapprochement d'un scénario de panne.

i) structuration de la trace en blocs

Pour rendre accessible la trace du raisonnement à un utilisateur, le système structure cette trace en blocs représentant chacun une portion de raisonnement focalisée sur un objet particulier du domaine.

Cette structuration en blocs offre des facilités d'explication en présentant un bloc complet qui constitue une partie aisément compréhensible de l'ensemble du raisonnement. Ainsi, sur l'exemple de la figure II.15, le bloc 1 est caractérisé par l'objet « console », le bloc 2 par le voyant d'alimentation « v1 », etc.

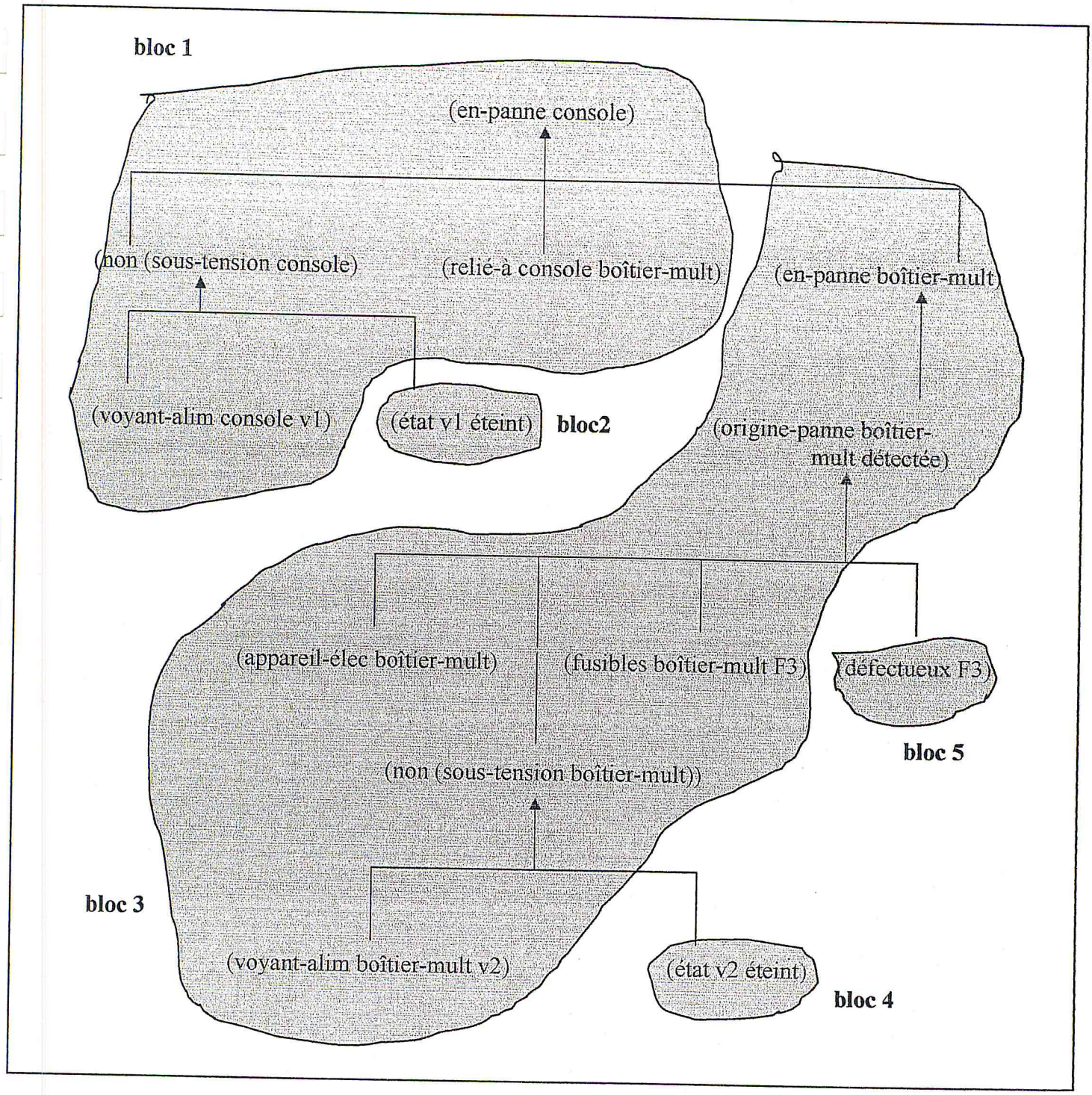


Figure II.15 - Mise en évidence des « blocs objets » du raisonnement

Pour expliquer à un utilisateur comment un fait donné a été déduit, le système linéarise l'arborescence de déduction de ce fait F en procédant comme suit : i) il garde tous les faits du bloc correspondant à l'objet du raisonnement de F, ii) il ne conserve dans les autres blocs que les faits liés directement a F.

La sous-arborescence obtenue pour le fait (en-panne console) est représentée dans la figure II.16.

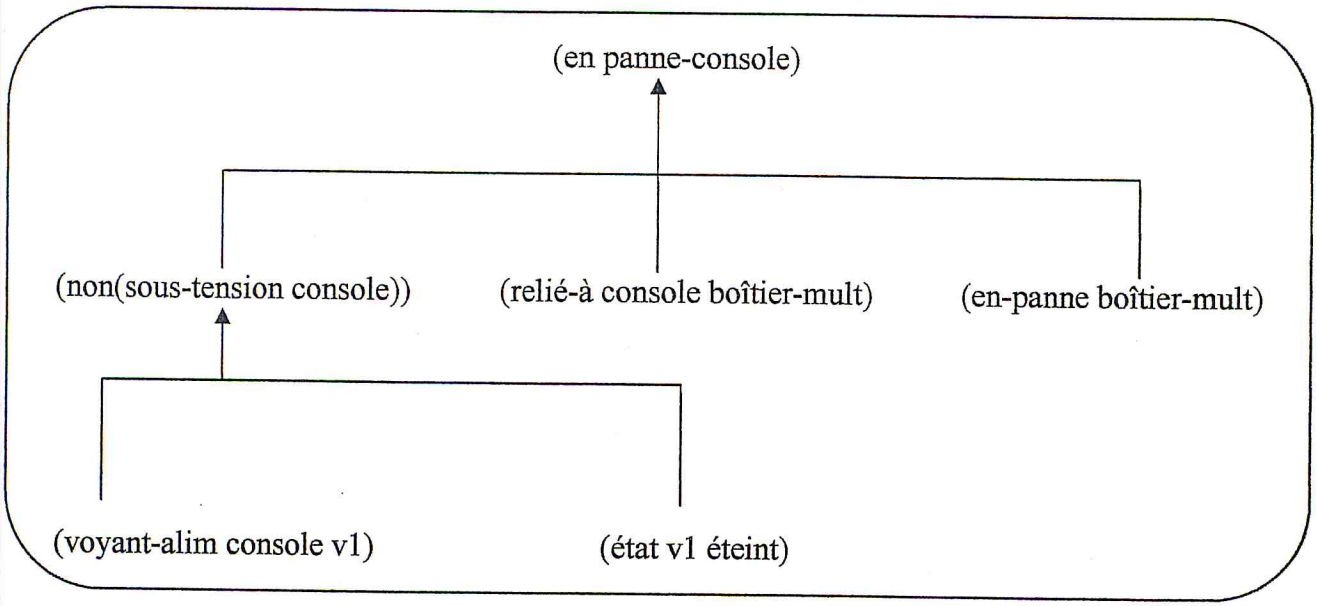


Figure II.16 - Sous-arborescence obtenue pour le fait (en-panne console)

On obtient alors l'explication suivante :

Console
voyant-alim : v1
→ état : éteint
sous-tension : faux
relié-a : boîtier-multiprise
→ en-panne : vrai

ii) recherche des faits pertinents

La notion de pertinence qui a été adoptée par C.Q.F.E suppose que l'utilisateur est un habitué du système et qu'il est non pertinent de lui représenter les faits qui se répètent d'une session à l'autre ainsi que les faits qui correspondent aux connaissances descriptives du domaine.

Pour continuer à simplifier l'explication obtenue après l'extraction de la sous-arborescence de déduction du fait, le système élimine de cette sous-arborescence les faits jugés non-pertinents. Il supprime pour chaque bloc, d'une part les faits fournis par l'utilisateur, et d'autre part, les faits du modèle conceptuel ou ceux qui peuvent être déduits en utilisant l'héritage des propriétés.

Ainsi, connaissant les relations suivantes :

- *la console est reliée au clavier*
- *la versatec est une imprimante*
- *tout appareil électrique est relié électriquement à la terre*
- *toute imprimante est un appareil électrique*

C.Q.F.E. élimine les faits contenus ou déductibles dans le modèle conceptuel et ne retient que les faits du diagramme de droite de la figure II.17 :

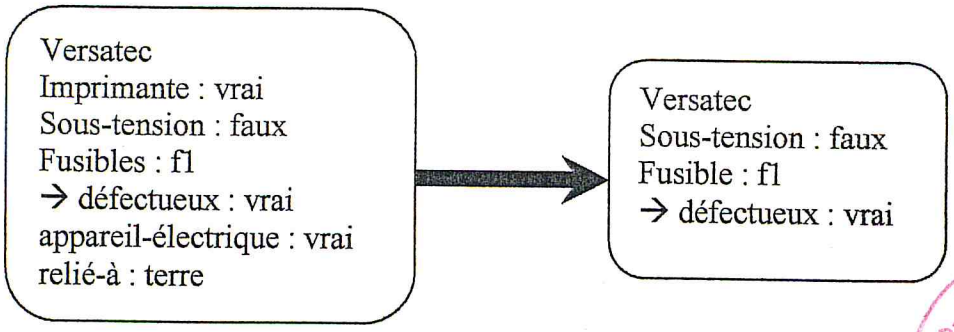


Figure II.17 - Faits retenus comme pertinents

iii) Rapprochement d'un scénario de panne

A partir des deux premières étapes, il s'agit de reconnaître parmi les objets du raisonnement, ceux dont le fonctionnement est anormal ou atypique, en s'aidant des connaissances du modèle conceptuel. Cette étape consiste à comprendre et expliciter la nature des liens entre les blocs d'objets constituant le raisonnement.

L'analyse d'arborescences de déduction fournies par le système expert a montré que les liens entre les différents blocs pouvaient être vus comme autant d'événements élémentaires permettant de reconstituer le scénario de la panne diagnostiquée. Il devient alors aisé d'indiquer le composant qui est à l'origine de la panne et d'intégrer le raisonnement dans un scénario explicatif tel que :

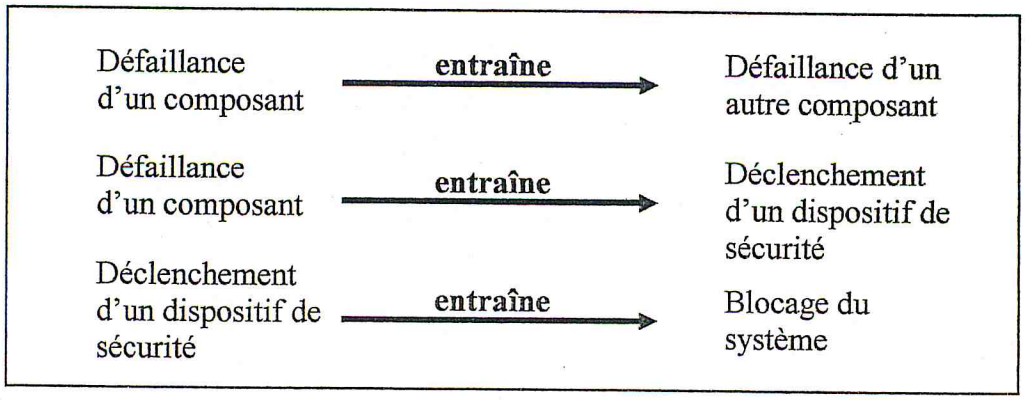


Figure II.18 - Scénario explicatif

L'approche développée par G. Kassel montre qu'il est possible de raisonner à partir de la trace de raisonnement. Il a été amené d'une part à structurer son explication et d'autre part à concevoir un modèle qui permette de différencier les faits pertinents de ceux qui ne le sont pas.

5.2.2. POURQUOI-PAS ?

5.2.2.1. Introduction

Safar [87] considère qu'un système expert comme MYCIN est beaucoup trop bavard lorsqu'il fournit des explications et que la masse d'informations contenue dans ces explications risque de noyer l'utilisateur sans lui apporter les éclaircissements espérés. Les idées directrices de son travail sont, d'une part, de résumer la trace du raisonnement suivi par le système pour éviter de noyer les concepts pertinents parmi les détails et, d'autre part, de développer des explications « négatives » pour montrer à l'utilisateur qui attend un résultat et en obtient un autre, quelles sont les différences entre son propre raisonnement et celui du système.

Le module d'explication répond aux questions des utilisateurs sur la valeur d'un attribut présent dans la base de faits après une session. Le système engendre deux types d'explications en réponse aux deux types de questions suivantes :

- *explication positive* : POURQUOI tel attribut a-t-il telle valeur ?
- *explication négative* : Pourquoi tel attribut N'a-t-il PAS telle valeur ?

Dans les deux cas, les explications fournies intègrent la totalité du raisonnement :

- en explication positive, le système fait ressortir les grandes lignes du raisonnement et les faits les plus significatifs,
- en explication négative, il choisit parmi les enchaînements invalides qui auraient pu conduire au résultat suggéré par l'utilisateur, l'enchaînement le plus proche du contexte décrit par la base de faits et souligne dans celui-ci les principales hypothèses non vérifiées qui lui ont interdit de conclure ce que l'utilisateur attendait.

Le système POURQUOI-PAS ? a été développé sur le moteur d'inférences ZERO+ associé à une expertise décrivant les principes d'enchères au Bridge.

5.2.2.2. Le module d'explication

Pour pouvoir répondre a une question de l'utilisateur le module d'explication du système POURQUOI-PAS ? opère en deux temps en appliquant la démarche suivante :

- rechercher le raisonnement invalide et les termes sur lesquels doit porter l'explication,
- mettre en forme l'explication de ces termes.

Pour répondre à une question de l'utilisateur sur le fait F, le système POURQUOI-PAS ? construit l'arborescence de déduction potentielle de F. Cette arborescence est ensuite élaguée pour ne conserver que les faits pertinents qui sont des termes-conditions ou conclusion de règles.

Au lieu de chercher le bon raisonnement (une branche de l'arbre de dérivation) sur la totalité de l'arbre, Safar propose de développer directement la branche pertinente en élaguant l'arbre au fur et à mesure de sa construction.

Le système procède de façon récursive en suivant les deux étapes suivantes :

i) considérer les règles qui auraient pu conclure sur le fait F cherché (dites aussi règles N-candidates),
ii) comparer leurs conditions invalides.

Un certain nombre de conditions invalides pertinentes seront sélectionnées et considérées comme les nouveaux faits cherchés dans l'étape de récursion suivante.

Illustrons cette approche sur un exemple. Supposons que l'on cherche pourquoi l'on n'a pas obtenu le fait F. Supposons que l'arbre de déduction du fait F, représentant les différents enchaînements de règles qui auraient pu conclure sur F, soit celui représenté en figure II.19.

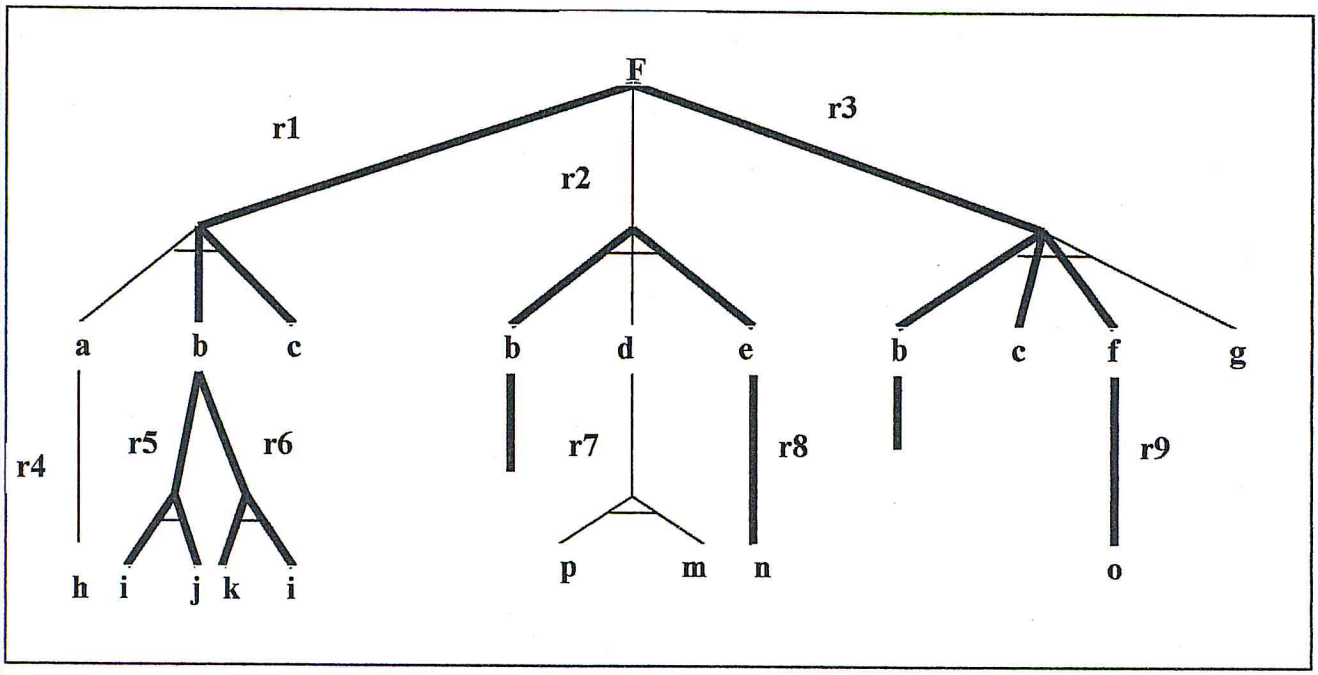


Figure II.19 - Arbre de dérivation du fait F

Les branches de l'arbre en gras représentent les conditions ou les règles vérifiées par la base de faits. Tout ce qui est en trait fin est invalide.

La figure II.20 montre la construction récursive du raisonnement recherché. Les branches de l'arbre en gras sont supprimées puisqu'elles représentent les conditions ou règles vérifiées par la base de faits donc inutiles pour cette phase.

Pour dégager les termes pertinents parmi tous les termes-conditions invalides, le système procède de la façon suivante :

i) s'il trouve des termes invalides commun à toutes les règles qui concluent sur le fait attendu, il les considère comme pertinents,

ii) sinon, il essaie de se ramener au cas précédent d'une part, en choisissant un sous-ensemble de règles invalides ayant en commun des termes invalides et d'autre part, en

essayant d'éliminer les autres règles suivant des critères « contextuels » (le qualificatif « contextuel » est associé par l'expert à certains attributs). Si l'un de ces derniers constitue un terme invalide dans une règle, cette règle est jugée trop éloignée du contexte initial de la base de faits. Les règles dites hors-contexte pour une base de faits donnée sont jugées trivialement invalides par le système et donc éliminable.

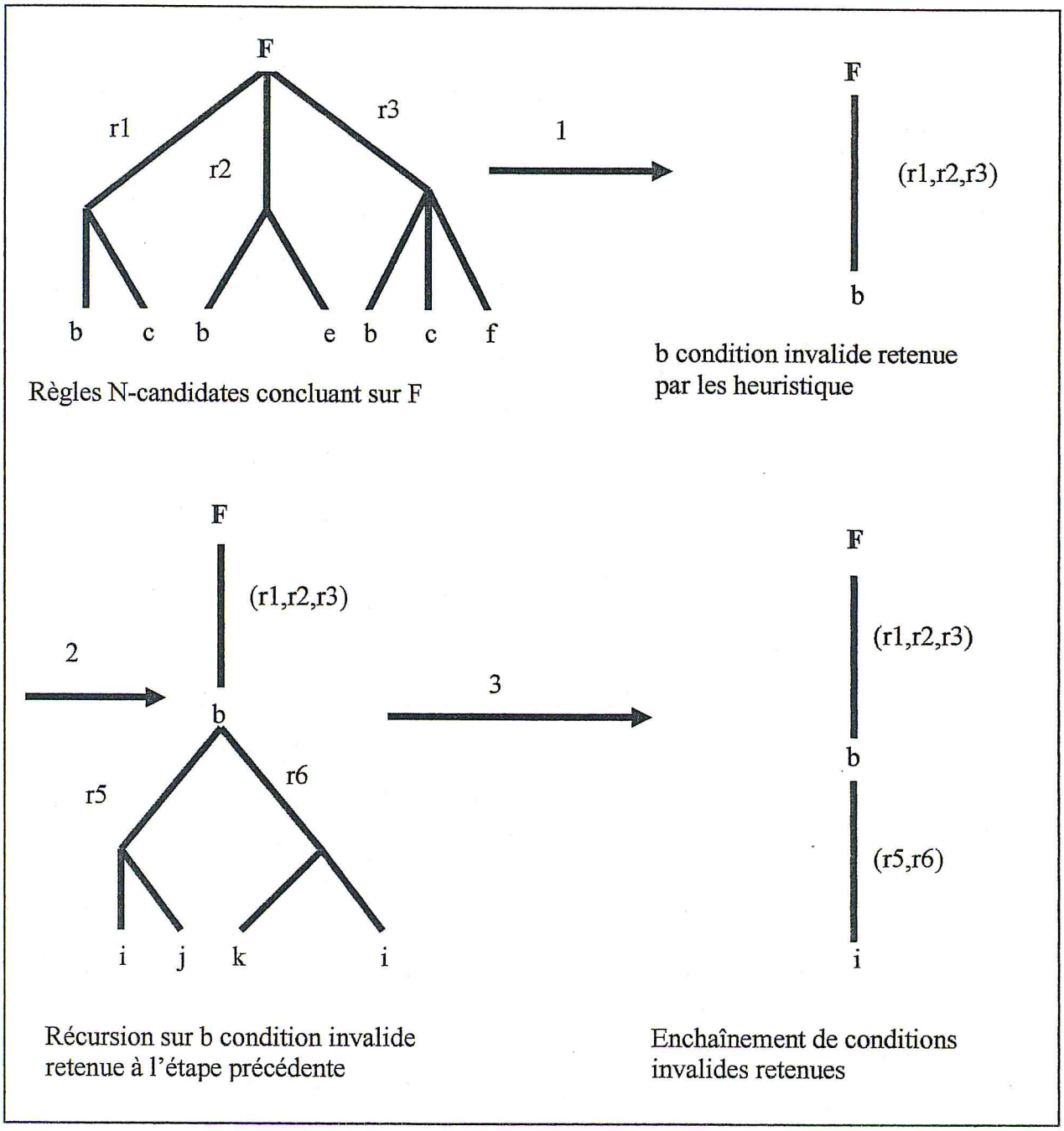


Figure II.20 - Les différentes étapes pour la recherche de la branche pertinente

5.3. Deuxième direction : modélisation du raisonnement explicatif
[Saïdi 92] [Lemaire 92]

La seconde approche est complémentaire et consiste à travailler au niveau de la production de l'explication. La première direction possède en effet des limitations intrinsèques qu'une représentation de qualité ne pourra jamais dépasser. Il ne suffit pas de posséder la « matière première » de l'explication, il faut aussi en extraire les éléments pertinents, la mettre en forme, l'adapter au contexte, choisir entre différentes explications potentielles. Il y a donc là un grand nombre de tâches à accomplir pour transformer une représentation d'un contenu quelconque en une explication que l'utilisateur acceptera. C'est l'aspect opératoire de l'explication qui fait référence à l'explication *en tant qu'objet à transmettre à l'utilisateur.*

De plus, il s'agit de rendre dynamique une représentation qui est par nature statique, c'est-à-dire de faire d'une trace de raisonnements ou d'une hiérarchie de concepts, un objet utilisable dans une interaction avec l'utilisateur. Plusieurs chercheurs ont montré que l'explication n'est pas seulement un objet textuel ou graphique, mais aussi ce qui émerge de l'interaction avec l'utilisateur. Il faudra donc accepter les interruptions de l'utilisateur en cours d'explication, savoir interpréter sa question, pouvoir revenir sur une explication déjà donnée et gérer un dialogue avec cet utilisateur. C'est l'aspect interactif de l'explication qui fait référence à l'explication *en tant que ce qui émerge du dialogue avec l'utilisateur.*

Les travaux de cette direction considèrent donc l'explication comme étant *une tâche de résolution de problèmes à part entière.*

5.3.1. TEXT

Mckeown [85], à partir d'analyse de textes, a reconnu l'existence de schémas standards reflétant la structure et le contenu d'une explication. Pour lui, chaque schéma permet d'atteindre un but du discours et est composé d'une suite d'unités de base qu'il appelle les *prédicats rhétoriques.*

Mckeown a identifié quatre schémas :

- *Identification Schema* pour fournir des définitions,
- *Constituency Schema* pour décrire une entité ou un événement par parties,
- *Attributive Schema* pour illustrer un point particulier d'un concept,
- *Contrastive Schema* pour décrire un objet en mettant en contraste un point important et un point négatif.

Dans la figure suivante, on trouve la description du *constituency schema.*

Constituency
Cause-effect* | Attributive* |
{ Depth-identification | Depth-attributive
{ particular-illustration | Evidence}
{ Comparison | Analogy}}+
{Amplification | Explanation | Attributive | analogy}

Figure II.21 : Constituency Schema de Mckoewn

*Lire « {} »= optionnel, « | »= alternative, « + » indique que l'item apparaît de 1 à n fois, « * » indique que l'item apparaît 0 à n fois*

Paris [88] a utilisé une combinaison des *constituency schemas* et de la trace pour engendrer des explications/descriptions de systèmes physiques. Elle a redéfini la structure d'un schéma et a ajouté *l'identification* comme premier item. Le schéma résultant est donné dans la figure II.22.

Identification (intro to superordinate)
If there is local expertise for superodinate, do **Process Trace** *of superordinate*
Constituency (description of subparts)
For each subpart, do:
If there is a local expertise do Depth **Identification** *else do* **Process Trace**
Attributive

Figure II.22 - Constituency Schema de Paris

Paris a aussi défini un schéma de trace qu'elle utilise en conjonction avec le *constituency schema.* Le processus de trace permet d'expliquer les processus fils et les descriptions optionnelles des parties évoquées.

Cette technique permet d'engendrer des textes cohérents à partir de connaissances qui peuvent être adaptées pour chaque utilisateur. Cependant, comme le souligne Paris[92], il n'est pas question pour cette technique de développer un véritable dialogue. En effet, les schémas ne donnent que l'objectif global du texte et non les sous-buts des différentes parties. L'information est compilée dans les schémas et n'est donc pas explicite. Or, lorsqu'un utilisateur pose une question sur une réponse donnée, elle concerne le plus souvent une partie du texte et non le texte complet. Pour répondre à une telle question, le système doit pouvoir comprendre le texte qu'il a produit et savoir quelles parties du texte n'ont pas atteint leur but. A l'aide de schémas, le système ne peut qu'engendrer un autre texte accomplissant l'objectif global.

5.3.2. EDGE

Cawsey [89] a également reconnu l'existence de formes récurrentes d'explications, mais propose de représenter ces structures du discours par une grammaire de règles, de façon à capturer l'organisation hiérarchique du discours. Dans le système EDGE, elle distingue deux processus de planification :

i) la planification du contenu d'explication,
ii) la planification de la représentation du texte à l'utilisateur.

Pour Cawsey, un texte explicatif n'est pas seulement une suite de propositions connectées entre elles. Des résumés, des entêtes, des introductions et même des méta-commentaires sont utilisés pour aider l'utilisateur à la compréhension du texte.

Cawsey reconnaît l'existence d'une certaine flexibilité dans les schémas en même temps, elle observe une ambiguïté dans les classifications. Elle a aussi constaté qu'aucun des schémas rhétoriques de Mckeown, qui se veulent pourtant génériques, ne permet de représenter certaines structures de discours propres à son domaine des circuits électroniques.

Un extrait de cette grammaire est représenté dans la figure II.23

Les composants de chaque règle sont optionnels mais l'ordre doit être respecté. Cette représentation (en grammaire de règles) permet une description à différents niveaux de la structure.

EXPLAIN	→	Describe, EXPLAIN, Behaviour, Summarize
EXPLAIN	→	Type of circuit, Summarize
Descibe	→	Type of circuit, Function, Describe Components
Describe	→	Analogy
Describe component	→	Function, I/O Behaviour
Fonction	→	I/O Dependencies
Type of circuit	→	Circuit Type, Functional Type
EXPLAIN Behaviour	→	EXPLAIN Dependencies, EXPLAIN I/O
Summarize	→	Function, I/O Behaviour

Figure II.23 - Extrait de la grammaire d'explication

Chaque règle de la grammaire est écrite sous forme d'un opérateur de plan. Chaque opérateur (appelé stratégie) possède différents champs :

- *le nom de l'opérateur :* il décrit le concept à expliquer dans la plan,
- *les arguments :* ils représentent généralement les objets du domaine,
- *les préconditions :* elles représentent la liste de concepts prérequis,
- *les contraintes :* elles constituent les conditions pour la sélection des instances de plan,
- *les sous-buts :* ils contiennent les opérateurs modélisant la partie droite de la règle de la grammaire,
- *un pattern de texte :* il représente les concepts terminaux de l'arborescence. Leur concaténation permet de produire le discours explicatif final.

Une telle représentation permet à Cawsey de construire une arborescence de l'explication, ce que ne fait pas la méthode employée par Mckeown.

5.4. Combiner ces deux directions [Lemaire 92]

Ces deux directions de recherche sont donc complémentaires. En les combinant on cumule alors leurs avantages respectifs. La plus vaste entreprise en ce sens est le projet EES (Explainable Expert System) [Neches et al 84], débuté en 1984 à l'*Information Sciences Institute* de Los Angeles.

Au niveau de la première direction, ce projet se donnait comme objectif de prendre en compte les problèmes d'explication dès la conception du système expert. Cette approche est née d'un constat : l'adjonction d'un module d'explication à un système expert existant conduit souvent à remettre en cause la représentation des connaissances de ce dernier ; NEOMYCIN en est le meilleur exemple. L'idée est donc de penser à l'explication dès la phase d'acquisition des connaissances. Plus concrètement, il s'agit de spécifier au mieux le futur processus d'explication (la seconde direction de recherche), par exemple en imaginant les formes de questions auxquelles le système devra répondre [Swartout et al. 87], ce qui induira des contraintes sur le type de connaissances à représenter dans le système expert (première direction de recherche).

Au niveau de la seconde direction, les travaux de Moore [Moore et al 89] et Paris [Paris et al 88] au sein de EES ont tiré parti de la présentation des connaissances du système expert pour concevoir un module d'explication performant, tenant compte de l'utilisateur et du contexte dynamique pour produire son explication.

5.4.1. Explainable Expert System (EES)

Swartout a généralisé la démarche développée avec XPLAIN dans le cadre du projet EES « EXPLAINable Expert System » [Neches & al 84], le principe général consistant à engendrer automatiquement un système explicateur est conservé. Deux modifications importantes ont été opérées et ont donné une nouvelle architecture du système (voir figure II.24) :

i) l'ajout de nouvelles formes de connaissances (terminologie, intégration et optimisation),
ii) la distinction essentielle entre historique du développement et historique d'exécution : le générateur d'explication utilise le premier pour justifier les actions du système expert, le second pour restituer la réalisation de ces actions (méthodes).

Neches, Swartout & Moore [84] se sont intéressés plus particulièrement à la génération du texte explicatif. Ils proposent une novelle architecture des systèmes explicatifs qui engendre cette fois-ci le texte dynamiquement en fonction d'opérateurs de plans et d'heuristiques. Le discours est constitué au départ d'un élément conceptuel décrivant un contenu explicatif. L'application de sources de connaissances provoque le remplacement de l'élément conceptuel initial par des éléments de plus en plus précis jusqu'à l'apparition d'éléments concrets de discours.

Un certain nombre de techniques de génération de langage naturel ont été développées répondant ainsi aux besoins des utilisateurs. Parmi ces techniques, on peut citer l'approche qui permet la génération flexible de textes en tenant compte des interventions de l'utilisateur. Lorsqu'un système engendre une explication en réponse à une question utilisateur, il tente d'atteindre un but que l'on appelle *le but du discours.* Avec une seule stratégie d'explication, on n'atteint pas très souvent ce but. De ce fait, lorsque l'utilisateur indique qu'il n'a pas compris l'explication produite, le système doit être capable d'engendrer une nouvelle explication en utilisant une stratégie différente de celle déjà employée. Dans ce cas, le planificateur doit engendrer une nouvelle explication de la partie non comprise en utilisant une autre stratégie.

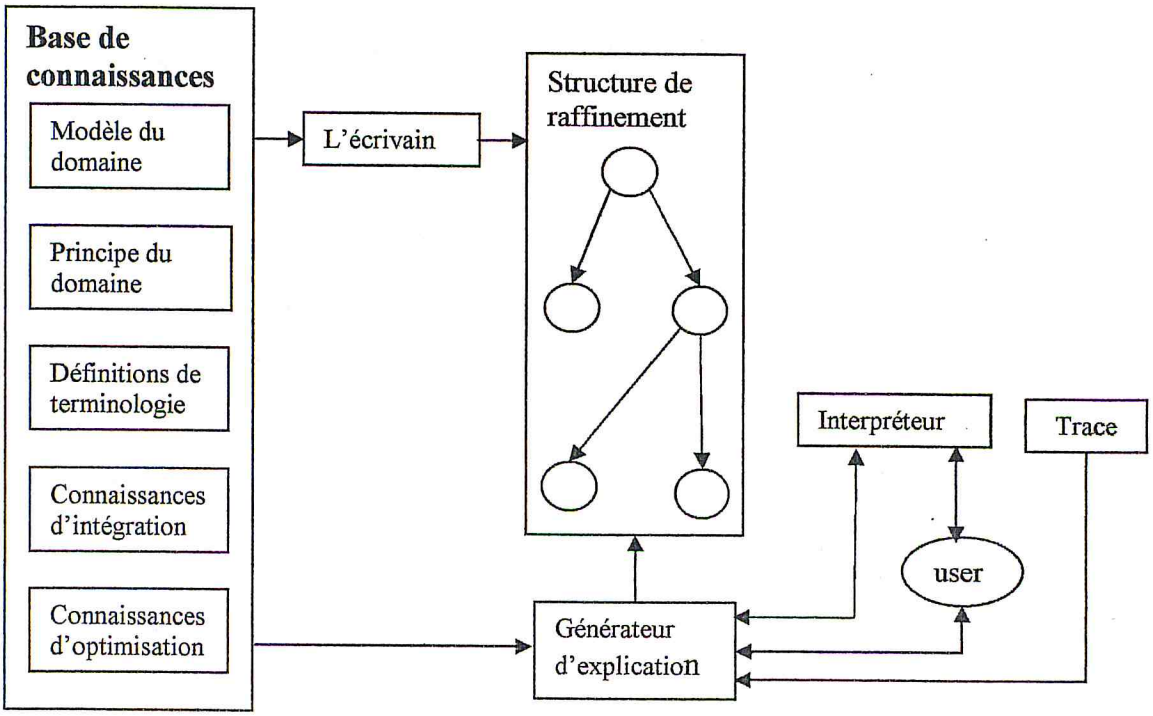


Figure II.24 : Architecture d'un Système Expert Explicable

Afin de pouvoir agir sur des parties du texte d'explication, il est nécessaire d'avoir explicité la structure intentionnelle du texte. Celle-ci consiste en une planification des buts/sous-buts auxquels on accroche des textes ou des morceaux de textes, qui peuvent être des phrases entières ou bien même des paragraphes. Un texte peut aussi être caractérisé en terme de relations de rhétorique entre des parties du même texte.

Ainsi, le planificateur de texte construit un plan détaillé de l'explication avant même de la générer. Ce plan explicite plus particulièrement la structure intentionnelle de l'explication, c'est-à-dire à la fois le but général de l'explication, les buts respectifs de chacune des parties de l'explication et les moyens rhétoriques utilisés pour les accomplir [Moore & al 89]. Ce plan d'explication est établi par un ensemble d'opérateurs de plan qui représente les stratégies d'explication.

Un opérateur de plan est composé de quatre parties :

i) *un effet* qui décrit le but de l'opérateur. Il peut être soit un but intentionnel (ex : persuader l'utilisateur a faire quelque chose), soit une relation rhétorique (ex : faire la différence entre deux choses),
ii) *une liste de contraintes* qui limitent l'applicabilité de l'opérateur. Elle peut faire référence au modèle de l'utilisateur ou à la base de connaissances,
iii) *un noyau* qui décrit le but principal du discours à présenter à l'utilisateur,
iv) *des satellites* ou sous-buts (souvent optionnels) qui permettent de renforcer le noyau.

Le noyau et le satellite ensemble renforcent l'effet de l'opérateur. Voici deux exemples d'opérateurs de plan :

EFFECT	**Persuade** the user to do an <action>
CONSTRAINTS	<goal> is a <domain goal>, and <action> is a step in acheving the <goal>, and system and user mutually believe, that <domain goal> is a goal of user.
NUCLEUS	**Motivate** the action in terms of the <domain goal>

Figure II.25 - Un opérateur de plan dont l'objectif est de persuader de faire une action

EFFECT	**Motivate** the <action> in terms of the <domain goal>.
CONSTRAINTS	<domain goal> is <goal> of the system, and <action> is a step towards achieving the goal.
NUCLEUS	**Inform** the user of the <domain goal>
SATELLITES	Tell the user that <action> is a **means** for achieving the <domain goal>

Figure II.26 - Un opérateur de plan

Les auteurs définissent deux types d'opérateurs : les opérateurs qui permettent d'atteindre le but intentionnel et ceux qui relient les bouts du texte explicatif.

Le système dispose de trois ensembles d'heuristiques :

i) les heuristiques de levée d'ambiguïté appelées *disambiguated heuristics*, dont le rôle est d'essayer de détecter la partie du texte que l'utilisateur n'a pas comprise,
ii) les heuristiques de sélection de plan appelées *plan selection heuristics*, dont le rôle est de choisir la stratégie de l'explication parmi les diverses stratégies possibles,
iii) les heuristiques de recouvrement dites *recovery heuristics*, dont le rôle est de choisir une stratégie alternative lorsque l'utilisateur n'a pas compris une explication.

Cette approche a été appliquée au Program Enhancement Advicer (PEA), un programme qui avait comme objectif d'améliorer les caractéristiques d'un programme utilisateur écrit en lisp [Neches & al 84]. Le principe consiste à vérifier le programme utilisateur pour faire des propositions de modification des instructions jugées inadéquates de son programme, l'objectif étant d'offrir une meilleure lisibilité du programme et/ou une maintenance facile. Nous allons illustrer cela sur un exemple tiré de [Moore & al 89].

S : you should replace (setq x 1) with (setf x 1).
U: why
S: i'm trying to enchance the maintenability of the program by applying transformation that enhance maintenability.

SETQ-TO-SETF is a trnsformation that enhances maintenability.

Dans cet exemple, le but intentionnel de l'explication est de persuader l'utilisateur d'appliquer l'action SETQ-TO-SETF en l'informant que le but *enhance maintenability* peut être atteint en exécutant les actions proposées. Si l'utilisateur pose ensuite la question *pourquoi ?* , le système doit chercher les raisons de l'échec de cette explication. A cet effet, il utilise des heuristiques pour déterminer la partie à remplacer. Dans ce cas, la question est interprétée comme «*why* SETQ-TO-SETF *is a transformation that enhances maintenability ?* ».

La réponse qui est engendrée est:

U: why ?
S: Transformation whose right-hand-side's use is more general than its self-hand-side's use.

SETF can be used to assign a value to any generalized-variable.
SETQ can only used to assign a value to a simple-variable.

D'autres travaux [Moore & al. 89] [Paris et al 88] plus intéressants, suggérant des plans de discours ont été proposés. Ils possèdent l'avantage de pouvoir représenter explicitement le raisonnement explicatif, cependant les discours proposés sont assez limités et le choix de ce qui va être dit ne peut pas être fait dynamiquement.

L'approche développée dans EES possède un avantage par rapport à celle de type grammaire : la maintenance ; en effet, dans la grammaire de Cawsey, les préconditions interdisent tous les conflits, c'est-à-dire qu'à tout moment, seule une règle est applicable. Avec cette approche, on a une meilleure séparation du contrôle des opérateurs par les heuristiques et les opérateurs de plans peuvent être conflictuels ; l'ajout de nouveaux opérateurs devient plus aisé.

6. Conclusion

Dans ce chapitre nous avons abordé le problème de l'explication dans les systèmes à base de connaissances. Nous avons donné les différentes acceptions du terme « explication » qui correspondent à des problématiques de recherche diverses.

Nous avons montré en décrivant les systèmes MYCIN, NEOMYCIN et XPLAIN, comment les techniques d'explication ont évolué, partant de la simple de notion de transparence, pour fournir progressivement au système expert un recul vis-à-vis de son raisonnement.

Différents travaux entrepris dans ce domaine ont été présentés. Nous les avons regroupé suivant deux directions :

- *la structuration de la connaissance et l'introspection.* Dans cette catégorie, nous avons décrit deux systèmes : le système C.Q.F.E d Kassel qui ne justifie pas les différentes étapes du raisonnement, mais présente une vue d'ensemble de ce raisonnement ; le système POURQUOI-PAS ? de Safar conçu pour fournir des explications après une session d'un système expert d'ordre 0+.
- *La modélisation du raisonnement explicatif.* Dans cet axe, deux systèmes ont été présentés : le système TEXT de McKeown qui est un système à base de schémas ; le système EDGE de Cawsey qui est un système à base de grammaires.

Une troisième direction consiste à combiner ces deux axes. Pour l'illustrer nous avons étudié le système EES de Swartout qui est un système à base de plans.

Enfin, ce chapitre nous a permis d'acquérir les notions essentielles sur l'explication dans les systèmes à base de connaissances. Elles seront ainsi implémentées dans le système SDPTV décrit dans le chapitre suivant.

1. Introduction

SDPTV (Système de Diagnostic de Pannes de TéléVisions) est un système à base de connaissances dont les objectifs sont d'assister un technicien en maintenance de télévisions à détecter les différentes pannes et de fournir des explications sur le raisonnement poursuivi.

Nous commençons par donner l'architecture du système SDPTV qui est composée d'un système expert qui s'occupe du diagnostic et d'un module d'explication chargé de la construction et de la transmission d'explications à l'utilisateur.

Pour la partie système expert, nous décrivons les représentations de connaissances utilisées (interne et externe) ainsi que le fonctionnement du moteur d'inférences (le chaînage avant, le chaînage arrière et le chaînage mixte) et du module de validation des connaissances.

Quant au module d'explication, nous présentons les différents types d'explications que celui-ci peut fournir, ainsi que les diverses catégories d'utilisateurs prises en compte lors de la construction des explications.

Enfin, nous terminons par décrire la maquette réalisée du système SDPTV.

2. Architecture du système SDPTV

SDPTV (Système de Diagnostic de Pannes de TéléVision) est un système expert explicatif (S.E.E). Il est composé de deux parties essentielles : d'une part un système expert (S.E) qui s'occupe de la résolution, c'est-à-dire du diagnostic, et d'autre part d'un module d'explication qui se charge de la justification du bien fondé des diagnostics établis par le système expert. La figure suivante résume l'architecture du système SDPTV.

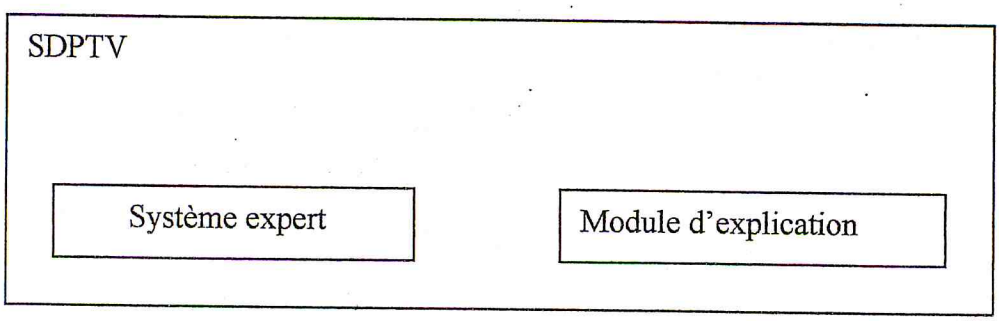


Figure III.1 - Architecture du système SDPTV

Cette architecture souligne le fait suivant : la qualité des explications fournies par le S.E.E dépend tout à la fois de l'architecture du S.E, en particulier la façon dont les connaissances sont représentées, et du module d'explication, c'est-à-dire la façon dont elles sont produites.

Concernant l'adéquation du S.E pour l'explication, voyons les hypothèses qui ont été formulées a posteriori par les auteurs de MYCIN :

Chapitre III

Conception et réalisation du système SDPTV

- à propos des règles de productions : le raisonnement sous-jacent, un enchaînement de modus ponens, est simple à comprendre ; d'autant que le vocabulaire de l'expert est utilisé dans la formulation des règles.
- à propos de son architecture : il n'est pas nécessaire que le S.E soit basé sur un modèle du raisonnement psychologiquement valide, c'est-à-dire qu'il modélise de prés la façon dont l'expert raisonne. Il suffit que le S.E utilise des connaissances semblables à celles de l'expert et qu'il pose ses questions en des termes identiques.

Le système SDPTV comporte :

- une base de connaissance,
- un moteur d'inférence d'ordre 0+ pour l'exploitation des connaissances,
- un module de validation des connaissances
- un module d'explication,
- une interface utilisateur conviviale.

3. Conception du Système Expert

3.1. Représentation des connaissances

3.1.1. Représentation externe

Le langage d'expression des connaissances est le pont de communication entre le système et l'expert. Dans notre système ce langage est fondamentalement bâti sur la logique propositionnelle d'ordre 0+.Les différentes connaissances exprimées dans ce langage sont relatives au domaine et au problème à résoudre.

Connaissances sur le domaine : elles sont de nature déductive et mise sous forme de règles de production dont la syntaxe sera détaillée dans cette partie.

Connaissances sur le problème : elles sont relatives à l'énoncé du problème, au but ou au fait à établir. Elles sont de même nature que les connaissances du domaine et exprimable dans le même langage.

La base de connaissances est composée de :

- Une base de règles : les connaissances y sont exprimées sous forme de règles de production,
- Une base de faits : les faits sont des couples (attribut, valeur),
- Un dictionnaire des faits : il contient des attributs.

i) Syntaxe du langage

La grammaire du langage que l'on doit respecter scrupuleusement est donnée ci – dessous :

<Base_Règle> = **DBR** <Liste_Règle> **FBR**;

<Liste_Règle> = <Règle> <Liste_Règle> / <Règle> ;

<Règle> = **DR** <Nom_Règle>

PRIO <PRIO_REG>

si <prémisse> **alors** <conséquent>

<EXPLICATION> **FR**

<prémisse > = <Condition> / <Condition> <Sép_cond> <Prémisse> **;**

(<Condition>) / (<Condition>) <Sép_cond> <Prémisse> **;**

<Sép_cond> = **ET** ;

<condition> = <Attribut_Numérique> <Comp_Num> <Attribut_Nummérique>/

<Attribut_Numérique> <Comp_Num> <Valeur_Nummérique>/

<Attribut_Alphanum> <Comp_Num> <Attribut_Alphanum>/

<Attribut_Alphanum> <Comp_Num> <Valeur_Alphanum>/

<Attribut_Alphanum> <Comp_Num> <Liste_Valeur_alphanum>/

<Attribut_Booléen> <Comp_Booléen> <Attribut_Booléen>/

<Attribut_Booléen> <Comp_Num> <Valeur_Booléen>/

<Comp_Num> = < / > / <= / >= / = / <> ;

<Comp_Alphanum> = <>/= ;

<Conséquent> = <Conclusion>/ <Conclusion> <Sép_concl> <Conséquent> ;

(<Conclusion>) / (<Conclusion>) <Sép_concl> <Conséquent> ;

<Sép_concl> = **ET** ;

<Conclusion> = <Attribut_Numérique> <Affectation> <Attribut_Nummérique>/

<Attribut_Numérique> < Affectation > <Valeur_Nummérique>/

<Attribut_Alphanum> < Affectation > <Attribut_Alphanum>/

<Attribut_Alphanum> < Affectation > <Valeur_Alphanum>/

<Attribut_Booléen> < Affectation > <Attribut_Booléen>/

<Attribut_Booléen> < Affectation > <Valeur_Booléen>/

<Attribut_Booléen> / **NON** <Attribut_Booléen>/ **NON**

(<attribut_booléen>) ;

<Valeur_Booléen> = VRAI / FAUX ;

<Base_Fait> = **DBF** <List_Fait> **FBF**;

<List_Fait> = <Fait> <List_Fait> / <Fait> ;

<Fait> = <Conclusion> / (<Conclusion>) ;

Exemple d'une règle :

DR REGLE004
PRIO 5
SI Micro_Contrôleur_Gestion = bloqué
Et Tension_Quartz = continue
ALORS Panne = Quartz_Micro_Contrôleur
EXPLICATION Si le microcontrôleur de gestion est bloqué Et la tension au bornes du quartz est continue (non alternative) Alors le quartz du microcontrôleur de gestion est défectueux. **FR**

Exemple d'un fait :

Tension_Alimentation_Secondaire > 12

ii) Sémantique du langage

La valeur d'un attribut est :

- val si le couple (attribut, val) appartient à la base de faits,
- non-obtenable si le couple (attribut, null) appartient à la base de faits.

Les attributs sont monovalués. S'il existe deux couples (attribut, val1) et (attribut, val2) avec val1≠val2, il y a détection d'une contradiction.

Evaluation d'une condition

La valeur de l'expression : attribut **comparateur** val :

- est **inconnue** si la valeur de l'attribut est Nil,
- résulte de la comparaison entre la valeur de l'attribut et val sinon.

Les comparateurs <,>, <=,>= ne s'appliquent qu'à des attributs de type numérique.

Les comparaisons avec les valeurs **vraies** et **fausses** ne s'appliquent qu'aux attributs de type booléen.

La valeur de l'expression : attribut1 **comparateur** attribut2 :

- est **inconnue** si un des deux attributs a une valeur Nil,
- résulte de la comparaison entre les valeurs de l'attribut1 et attribut2 sinon.

Evaluation d'une conclusion

L'évaluation d'une conclusion attribut1 = attribut2 provoque :

- l'ajout du couple (attribut1, val2) si le couple (attribut2,val2) appartient à la base des faits ,

- l'ajout du couple (attribut1, attribut2) à la liste à évaluer sinon le couple est en attente d'évaluation : dès qu'une valeur val2 est attribuée à attribut2, le couple est supprimé de la liste à évaluer et le couple (attribut1, val2) est ajouté à la base des faits.

L'évaluation d'une conclusion *attribut* est traitée comme *attribut* = **vrai** et **non** *attribut* (ou **pas** *attribut*) comme *attribut* = **faux**.

3.1.2. Représentation interne

Représenter les connaissances dans un ordinateur consiste à trouver une correspondance entre le monde extérieur et un système symbolique permettant de raisonner [Bonnet 84].

i) Structure interne d'une règle

L'ensemble des règles forme la base de règles du système : c'est le savoir-faire de l'expert. Une règle est composée de:

- Un nom de règle : il permet de l'identifier,
- Une priorité : un code pour désigner la priorité d'une règle,
- Nombre de conditions,
- Nombre de conclusions,
- Nombre des conditions non encore vérifiées,
- Etat de la règle : état actuel de la règle (bloquée ou non),
- Pointeur vers la liste des conditions,
- Pointeur vers la liste des conclusions,
- Pointeur vers la règle suivante : c'est une entrée vers une autre règle,

Nom règle
Code priorité
Nombre des conditions
Nombre des conclusions
Nombre des conditions non encore vérifiées
Etat de la règle
Pointeur vers la liste des conditions
Pointeur vers la liste des conclusions
Pointeur vers la règle suivante

Figure III.2 - Structure d'une règle

ii) Structure interne d'une condition

Une condition est représentée par :

- Opérande1 : désigne le premier opérande de la condition,
- Comparateur : désigne l'opérateur de comparaison (=, <, …),
- Opérande2 : désigne le deuxième opérande de la condition,
- Type condition : indique le type de comparaison :
  '0' : attribut - attribut,
  '1' : attribut - valeur,
- Etat : indique l'état de la condition :
  '-1' : non vérifiée,
  '0' : non encore évaluée,
  '1' : vérifiée,
- Pointeur vers la condition suivante : pointe vers la prochaine condition dans une règle.

Opérande 1
Comparateur
Opérande 2
Type de condition
Etat
Pointeur vers la condition suivante

Figure III.3 - Structure d'une condition

iii) Structure interne d'une conclusion

Une conclusion est composée de :

- Opérande 1 : désigne le premier opérande de la conclusion,
- Opérande 2 : désigne le deuxième opérande de la conclusion,
- Type affectation : indique le type d'affectation :
  '0' : attribut - attribut,
  '1' : attribut - valeur,
- Pointeur vers la conclusion suivante : pointe vers la prochaine conclusion dans une règle.

Etat de conclusion
Opérande 1
Type de conclusion
Opérande 2
Pointeur vers la conclusion suivante

Figure III.4 - Structure d'une conclusion

iv) Structure interne d'un attribut

Les Attributs représentent en quelque sorte les variables globales du système. Ils sont répertoriés dans le dictionnaire des faits.

Un Attribut est représenté par :

- Nom de l'Attribut : il permet d'identifier l'attribut,
- Type de l'Attribut : désigne le type de l'attribut (booléen, numérique ou alphanumérique),
- Une question sur l'attribut (dans le cas où il s'agit d'un attribut demandable). Son rôle est de rendre le dialogue avec l'utilisateur plus naturel.
- Pointeur vers la liste des règles où ce cet attribut est en conclusion
- Pointeur vers la liste des règles où cet attribut est en conclusion
- Pointeur vers l'attribut suivant : branchement vers la structure de l'attribut suivant.

Nom de l'Attribut
Type de l'Attribut
Question
Pointeur vers liste règles en condition
Pointeur vers liste des règles en conclusion
Pointeur vers l'attribut suivant

Figure III.5 - Structure d'un attribut

v) Compilation de la base de connaissances

Par cette représentation des connaissances, nous remarquons que les faits pointent les règles et inversement les règles pointent les faits, ainsi la base de connaissances est structurée en un véritable réseau que parcourt le moteur d'inférences. C'est la mise à jour et la constitution de ce réseau qui est appelée «compilation de la base de connaissances ».

La compilation préalable de la base de connaissances minimise le nombre de tests effectués à chaque cycle du moteur d'inférences, ceci permet un gain de temps considérable, il est très courant que plusieurs règles possèdent des prémisses ou des ensembles de prémisses communs. Il est donc naturel de ne faire qu'une seule fois les tests élémentaires sur ces prémisses pour l'ensemble de toutes les règles qui les contiennent.

Cette compilation nous permet aussi d'optimiser au maximum la représentation des connaissances en mémoire ce qui revient à dire qu'un attribut (ou fait ou règle) ne doit être présent qu'en un seul exemplaire dans le système.

3.2. Moteur d'inférences

Le moteur d'inférences du système possède les caractéristiques suivantes :

- d'ordre 0+.
- monotone.
- fonctionne en monde ouvert.
- son régime de fonctionnement est irrévocable.
- le chaînage est mixte combinant le chaînage avant et arrière.

3.2.1. Chaînage avant

Le fonctionnement du moteur en chaînage avant est une déduction de nouveaux faits à partir des faits contenus dans la base de faits.

Lors du processus de déduction, les seuls changements affectés à la base de faits sont des ajouts de nouveaux faits, cela est dû à l'hypothèse de monotonie sur laquelle repose notre moteur. Ces faits déduits sont les conclusions des règles déclenchées après satisfaction de leurs contraintes à partir des faits initiaux. La liste des règles où ce fait est en condition (voir III-2-2) permet de traiter uniquement les règles concernées.

Le cycle de base du chaînage avant est le suivant :

a) le *filtrage* : calcul de l'ensemble des règles déclenchables appelé aussi espace de conflits. Une règle est considérée comme étant déclenchable si toutes ses conditions sont vérifiées (nombre de conditions non vérifiées = 0).

b) la *résolution de conflits* : cette étape permet de déterminer la règle qui sera effectivement déclenchée. Il existe plusieurs stratégies de résolution de conflits. Pour notre système nous avons adopté la stratégie suivante :

- Choix da la règle ayant la plus grande priorité.
- S'il existe plusieurs règles ayant la même priorité alors le choix se fait selon l'ordre d'écriture des règles (ordre dans la base de règles)

c) le *déclenchement* : déclenchement de la règle choisie par l'étape précédente : ajout des faits en conclusion de la règle à la base de faits.
A la fin de cette phase, un test de cohérence de la base de faits est effectué pour déterminer si un fait ajouté est en contradiction avec un autre existant, alors le processus de résolution s'arrête et une erreur d'incohérence est signalée à l'utilisateur.

ALGORITHME DU CHAINAGE AVANT

Soit *BF* une base de faits, *BR* une base de règles et *F* le fait que l'on cherche à établir,
Calcul de l'espace de conflits : *EC* = Filtrage(*BR* , *BF)*
Tant que *F* n'est pas dans *BF* **et** *EC* non vide **boucler**
Choisir une règle applicable : R = Résolution_Conflits (*EC*)
Déclencher R (BF = BF union conclusions de R)
Désactiver R (BR = BR – R)

Calcul de l'espace de conflits : *EC* = Filtrage(*BR* , *BF)*
fin boucler
Si F est dans BF **alors**
F est établi
Sinon
F n'est pas établi
Fin si

Remarque : si l'on souhaite faire une saturation de la base de faits (c-à-d : déduire tout ce qui est déductible), il suffit de modifier l'algorithme précédent en enlevant la première condition d'arrêt (*F* n'est pas dans *BF*).

3.2.2. Chaînage arrière

Contrairement au chaînage avant qui est dirigé par les données, le chaînage arrière est dirigé par les buts.

L'utilisateur a la possibilité d'assigner au système un but hypothétique que ce dernier va s'efforcer de prouver grâce aux règles de BR et aux faits qu'il connaît déjà.

En chaînage arrière, le système recherche les règles qui concluent sur le but recherché et, pour chacune d'elles, il évalue ses prémisses suivant BF. Les prémisses qui ne sont pas vérifiées deviennent des sous-buts que le système va s'efforcer de prouver, en recherchant les règles susceptibles de les déduire, et ainsi de suite... . Le système construit ainsi l'arbre ET/OU de résolution du problème. Les nœuds « OU » représentent l'ensemble des règles permettant de conclure sur un fait, et les nœuds « et » représentent la conjonction des prémisses d'une règle qu'il faut vérifier pour prouver le but initial.
Si le système n'a pas suffisamment d'informations pour prouver le but recherché, il pose des questions à l'utilisateur.

Le cycle de base du chaînage arrière est constitué, comme celui du chaînage avant de trois phases :

a) le *filtrage* : l'ensemble de conflits est constitué des règles dont l'une des conclusions coïncide avec le but recherché.
Cet ensemble est construit, à chaque cycle, pour chaque sous-but qui n'est pas vérifié.

b) la *résolution de conflits* : cette étape permet de déterminer une règle parmi celles de l'ensemble de conflits. Il existe plusieurs stratégies de résolution de conflits. La stratégie adoptée est la même que la précédente (chaînage avant).

c) le *déclenchement* : cette phase est d'un intérêt particulier car elle se charge de la décomposition du but courant non vérifié, en un ensemble de sous-buts à vérifier. Dans le cas où le but courant est vérifié il fera partie de l'ensemble des faits établis.

ALGORITHME DU CHAINAGE ARRIERE

A chaque cycle le moteur construit un arbre de résolution ET/OU, à partir du but initial et l'explore en profondeur. Le cycle de fonctionnement est :

- Déterminer les faits de BF unifiables avec le but courant et l'ensemble des règles de BR où le but figure en conclusion :

1- Si les ensembles précédents sont vides, on revient à l'étape précédente
2- Si le but courant est unifiable avec au moins un fait de BF, on mémorise le reste des éléments de BF avec lesquels il est unifiable (nœud OU), ainsi que la liste des sous-buts en attente, et on reprend le raisonnement à partir de celle-ci.

- Choisir parmi les règles de l'étape précédente une règle (nœud OU de l'arbre)

1- Sauvegarder le contexte courant, à savoir la liste des règles restantes et la liste des sous-buts en attente.
2- Propager les conditions de la règle choisie, ainsi que la liste des sous-buts en attente.
3- Construire la nouvelle liste de sous-buts, à partir des prémisses précédentes (nœud ET) et le reste des sous-buts, et reprendre le raisonnement à partir de celle-ci.

Le moteur s'arrête lorsque la liste des sous-buts est vide. Dans ce cas, le problème est résolu ; ou bien il s'arrête après épuisement de toutes les possibilités de décompositions ou après détection d'une boucle (rencontre d'un but déjà démontré, ou déjà envisagé et menant à un échec). Dans ces deux derniers cas, le but initial ne peut être vérifié.

3.2.3. Chaînage mixte

Le chaînage mixte combine les deux chaînages précédents. Son algorithme est le suivant :

ALGORITHME DU CHAINAGE MIXTE

a) lancer le chaînage avant pour saturer la base de connaissances

b) **Tant que** la valeur de but est inconnue **et** le but est déductible **boucler**
- appliquer l'algorithme du chaînage arrière avec le paramètre d'appel but (ce qui conduit à un dialogue avec l'utilisateur, et peut être à l'ajout d'un nouvel élément à BF)

- appliquer l'algorithme du chaînage avant (de façon à tirer toutes les conséquences de l'apport d'information résultant du dialogue avec l'utilisateur)

fin boucler
Si F est dans BF **alors**
Indiquer à l'utilisateur la valeur de but
Sinon
Indiquer à l'utilisateur que le but ne peut être déduit
Fin si

Exemple :

Pour illustrer ce type de mécanisme, prenons les données suivantes :

Base de faits (BF) = { },

Base de règle (BR) = {

R1 : *Si* Prise_Secteur_Racordée = non
Alors Panne = Prise secteur non raccordée,

R2 : *Si* Interrupteur_Secteur_Enfoncé = non
Alors Panne = Interrupteur secteur non enfoncé,

R3 : *Si* Led_M/V = éteinte *Et* Tension_Alimentation_Secondaire = 0
Alors Panne = Interrupteur secteur défectueux
}

L'arbre ET/OU suivant récapitule le raisonnement suivi par le système pour arriver au diagnostic : Panne = Interrupteur secteur défectueux.

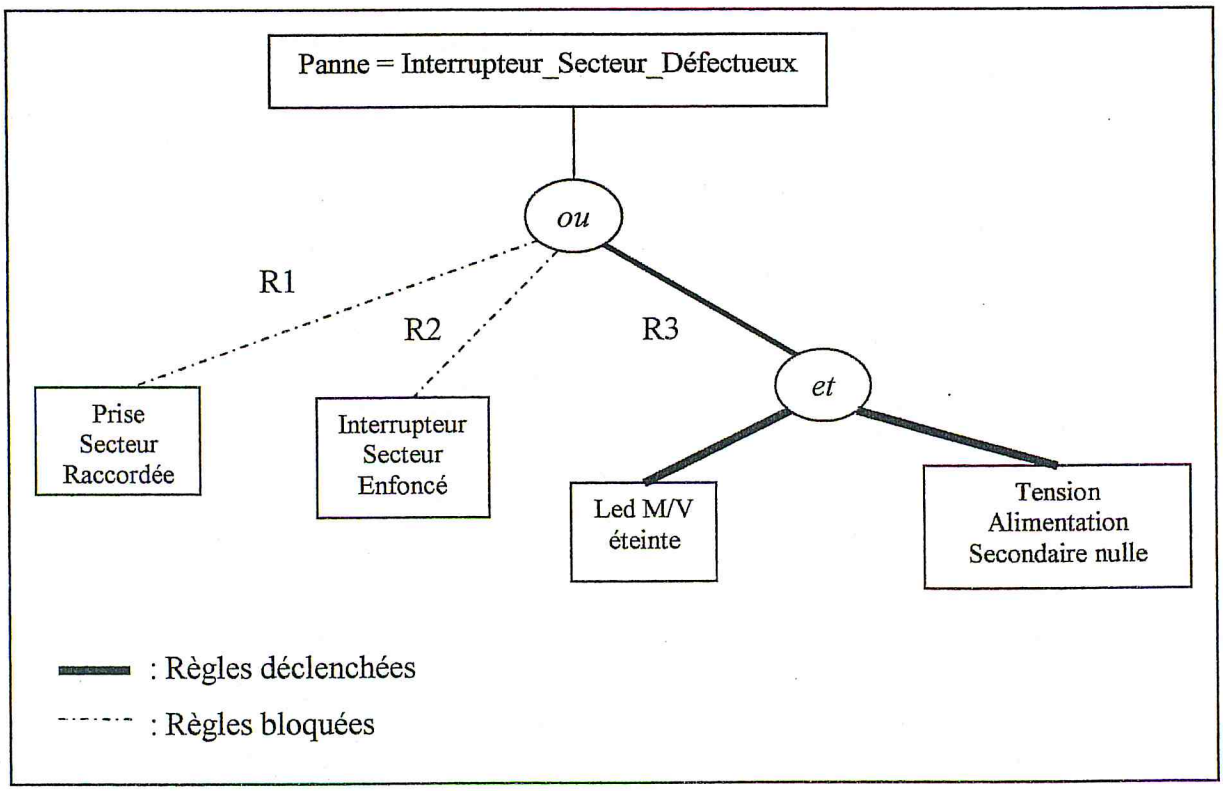


Figure III.6 - Arborescence de déduction du fait Panne

3.3. Module de validation des connaissances

Ce module est très important pour le bon fonctionnement du système. Il permet d'assurer la cohérence de la base de connaissances.

Le module de validation des connaissances est chargé en particulier de la détection de toute contradiction dans la base de faits. De ce fait, lors d'une application (déclenchement) d'une règle par le moteur d'inférences, celui-ci fait appel au module de validation pour vérifier si la conclusion de la règle est en contradiction avec la base de faits.

Exemples :

1) - le fait Led_M/V = éteinte apparient à BF
   - et R la règle en cours d'application conclue que Led_M/V = allumée
2) - le fait Tension_alim_THT > 115 apparient à BF
   - et R la règle en cours d'application conclue que Tension_alim_THT = 0

En cas de détection de conflits le processus de résolution est interrompu et une erreur est signalée à l'utilisateur.

4. Conception du Module d'Explication

L'objectif principal du système SDPTV est de fournir des explications sur son raisonnement.

Cette fonction d'explication est d'une grande importance. Dans certains cas l'utilisateur a besoin de savoir pourquoi ou comment tel résultat a été obtenu et non pas le résultat en lui même.

4.1. Différents types d'explication

Afin de rendre son raisonnement aussi transparent que possible, le système SDPTV propose deux types d'explications :

- Explications en cours de session
- Explications en fin de session

4.1.1. Explication en cours de session

En cours de session, l'utilisateur a la possibilité de poser des questions sur les inférences effectuées par le système du type : « Comment avez-vous obtenu tel résultat ? ».

Un deuxième type d'explication proposé est celui de la justification des questions posées à l'utilisateur.

Etant donné que le système fonctionne en chaînage mixte, il est souvent amené à poser des questions au cours du raisonnement. Ses questions sont parfois inattendues par l'utilisateur. Ce dernier pose alors des questions du genre « Pourquoi me posez-vous cette question ? ». Le système fait alors appel au module d'explication afin de générer une justification du bien fondé de la question posée.

a) Explications lors de la présentation de résultats intermédiaires

Lorsque le système fonctionne en mode pas à pas, la main est rendue à l'utilisateur après chaque application de règle. Le système affiche alors les conclusions de la règle appliquée qui viennent d'être ajoutées à la base de faits.

A ce moment, le système est capable de justifier si besoin est les résultats (intermédiaires) présentés à l'utilisateur. Pour ce faire, le module d'explication indique à l'utilisateur la dernière règle déclenchée qui a permis d'obtenir les résultats affichés.

Exemple : Pour illustrer ce type d'explication, nous allons prendre les données suivantes :

Base de faits (BF) = {A, B, C, H, F, D},

Base de règle (BR) = {

R2 : *Si* B *Et C Et H Alors* F
R7 : *Si* A *Et F Alors* D,
R11 : *Si* D *Et S Alors* G,
R17 : *Si* N *Et P Alors* G
}

L'état de l'arbre ET/OU produit par le système est le suivant :

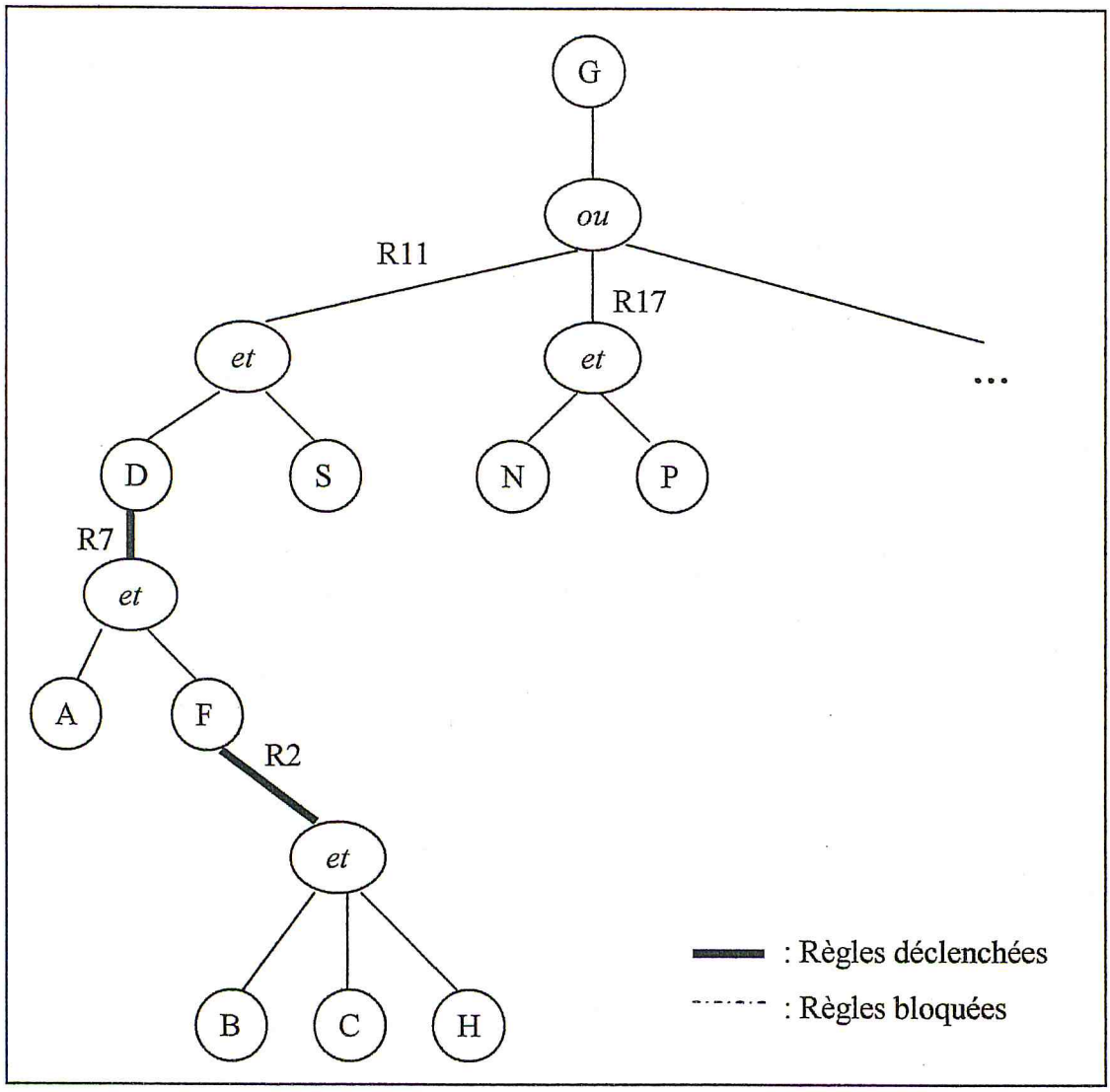


Figure III.7 - Partie de l'arbre de dérivation du fait G

La figure ci-dessous est un extrait de conversation en cours de session entre l'utilisateur et le système.

Système : J'ai déduit le fait D.

Utilisateur : Comment avez-vous déduit le fait D ?

Système : Je sais que :

- A
- F

J'ai donc utilisé la règle 'R7' :

[Si A
Et F
Alors D]

Qui m'a permis de conclure sur D.

Utilisateur : Comment avez-vous déduit le fait A ?

Système : Le fait A a été affirmé par l'utilisateur.

Utilisateur : Comment avez-vous déduit le fait F ?

Système : Je sais que :

- B
- C
- H

J'ai donc utilisé la règle 'R2'

[Si B
Et C
Et H
Alors F]

Qui m'a permis de conclure sur F.

Figure III.8 - Exemple d'explication de résultats intermédiaires en cours de session

b) Justification des questions posées à l'utilisateur

Lorsque le chaînage arrière est déclenché avec comme but un fait demandable et dont la valeur n'est pas connue, le système le pose comme question à l'utilisateur.

A ce niveau, il est possible que l'utilisateur ne comprenne pas ou désire savoir les raisons pour lesquelles le système lui pose cette question. Dans ce cas, le système fait appel au module d'explication pour justifier la question posée. Ainsi, le module d'explication affiche l'ensemble des règles représentant le chemin menant au but final recherché. Il commence par présenter la règle concluant sur le but final et termine par la règle dont l'une des conditions a été posée en question.

Ci-après, un exemple pour illustrer ce type d'explication.

Exemple : Reprenons l'arbre ET/OU de la figure III.7 :

BF = {},

Système : Quelle est la valeur de A ?

Utilisateur : Pourquoi me posez-vous cette question ?

Système : 1) Je cherche à établir le fait G.

Pour cela j'utilise la règle 'R11'

[Si D
Et S
Alors G]

2) Or, pour déclencher cette règle, j'ai besoin de connaître le fait D.

Pour déterminer le fait D, j'utilise la règle 'R7'

[Si A
Et F
Alors D]

3) Or, pour déclencher cette règle, j'ai besoin de connaître le fait A.

Le fait A étant demandable, je le pose comme question.

Figure III.9 - Exemple de justification d'une question posée à l'utilisateur

4.1.2. Explication en fin de session

Lorsque le système termine le raisonnement, nous offrons à l'utilisateur la possibilité de poser une question sur le résultat obtenu. En fin de session, deux types d'explication sont proposés : explication positive de type « pourquoi tel résultat ? » et explication négative du type « pourquoi pas tel autre résultat ? ».

a) Explications positives

Ce sont des explications répondant à des questions du type « Comment as-tu obtenu tel résultat ? ». Dés lors le système présente à l'utilisateur l'ensemble du raisonnement poursuivi en termes de règles et de faits et ce en utilisant l'arbre ET/OU et un certain nombre de connaissances utiles au raisonnement.

Exemple : Pour illustrer ce type d'explication, nous allons prendre les données suivantes :

Base de faits (BF) = {H, V, W, Q, non I, M, O, R, B},

Base de règle (BR) = {

R11 : *Si* ... *Alors* B,
R17 : *Si* ... *Alors* B,
R19 : *Si* V *Et* W *Alors* Q,
R24 : *Si* H *Et* Q *Et* R *Alors* B
R31 : *Si* I *Et* J *Alors* R,
R35 : *Si* M *Et* O *Alors* R,
}

L'état de l'arbre ET/OU produit par le système est le suivant :

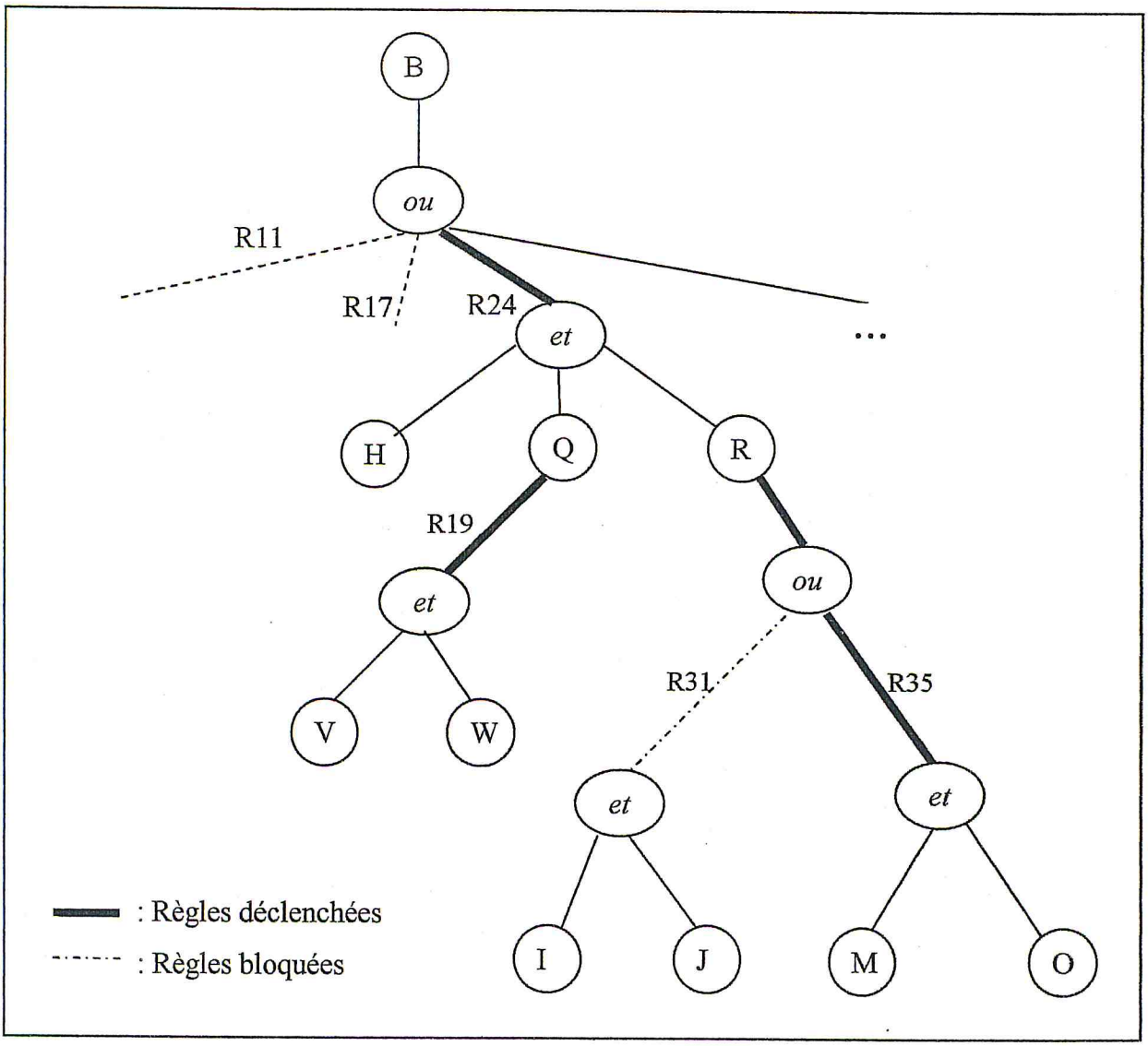


Figure III.10 - Partie de l'arborescence de déduction du fait B

La figure suivante illustre une explication en fin de session sur l'ensemble du raisonnement :

Système : J'ai déduit le résultat B.

Utilisateur : Comment avez-vous obtenu ce résultat ?

Système : 1) Je sais que :

- M (affirmé par l'utilisateur).
- O (affirmé par l'utilisateur).

Par conséquent, j'ai utilisé la règle 'R35'

[Si M
Et O
Alors R]

Pour établir le fait R.

2) Je sais que :

- V (affirmé par l'utilisateur)
- W (affirmé par l'utilisateur)

Par conséquent, j'ai utilisé la règle 'R19'

[Si V
Et W
Alors Q]

Pour établir le fait Q.

3) Je sais que :

- H (affirmé par l'utilisateur)
- Q (déduit par le système)
- R (déduit par le système)

Par conséquent, j'ai utilisé la règle 'R24'

[Si H
Et Q
Et R
Alors B]

Pour établir le fait B.

Figure III.11 - Exemple d'explication positive en fin de session

b) Explications négatives

Lorsque le résultat final est présenté à l'utilisateur, ce dernier peut être en désaccord avec le système, c'est-à-dire qu'il s'attendait à un résultat R1 autre que celui présenté. Il se pourrait aussi que l'utilisateur veuille savoir par curiosité pourquoi le résultat R1 n'a pas été obtenu. Pour s'en convaincre l'utilisateur à la possibilité de demander au système « Pourquoi n'avez-vous pas obtenu le résultat R1? ».

Pour répondre à cette question le système recherche les raisons pour lesquelles le résultat R1 n'a pu être obtenu.

Il existe deux cas de figure où le résultat R1 ne peut être atteint (ou n'a pas été atteint) :

- La ou les règles concluant sur R1 ont été écartées au cours du raisonnement.
- La ou les règles concluant sur R1 sont en attente de déclenchement.

i) Règles écartées : règles N-candidates

Dans ce cas, le module d'explication recherche toutes les connaissances qui ont permis d'écarter les règles dont le but est R1. Il explore, a cet effet, l'arbre ET/OU, et dégage l'ensemble des règles N-candiadtes.

Si le nombre des règles N-candidates est égal à 1, le système se contente d'afficher la ou les conditions non vérifiées (invalides) de la règle écartée.

Exemple : Pour illustrer ce type d'explication, nous allons prendre les données suivantes :

Base de faits (BF) = {non D, N, B, A, Résultat = Z},

Base de règle (BR) = {

R75 : *Si* V *Et* M *Alors* Résultat = T,
R90 : *Si* D *Et* S *Alors* Résultat = Y,
R99 : *Si* N *Et* B *Et* A *Alors* Résultat = Z
}

L'état de l'arbre ET/OU produit par le système est le suivant :

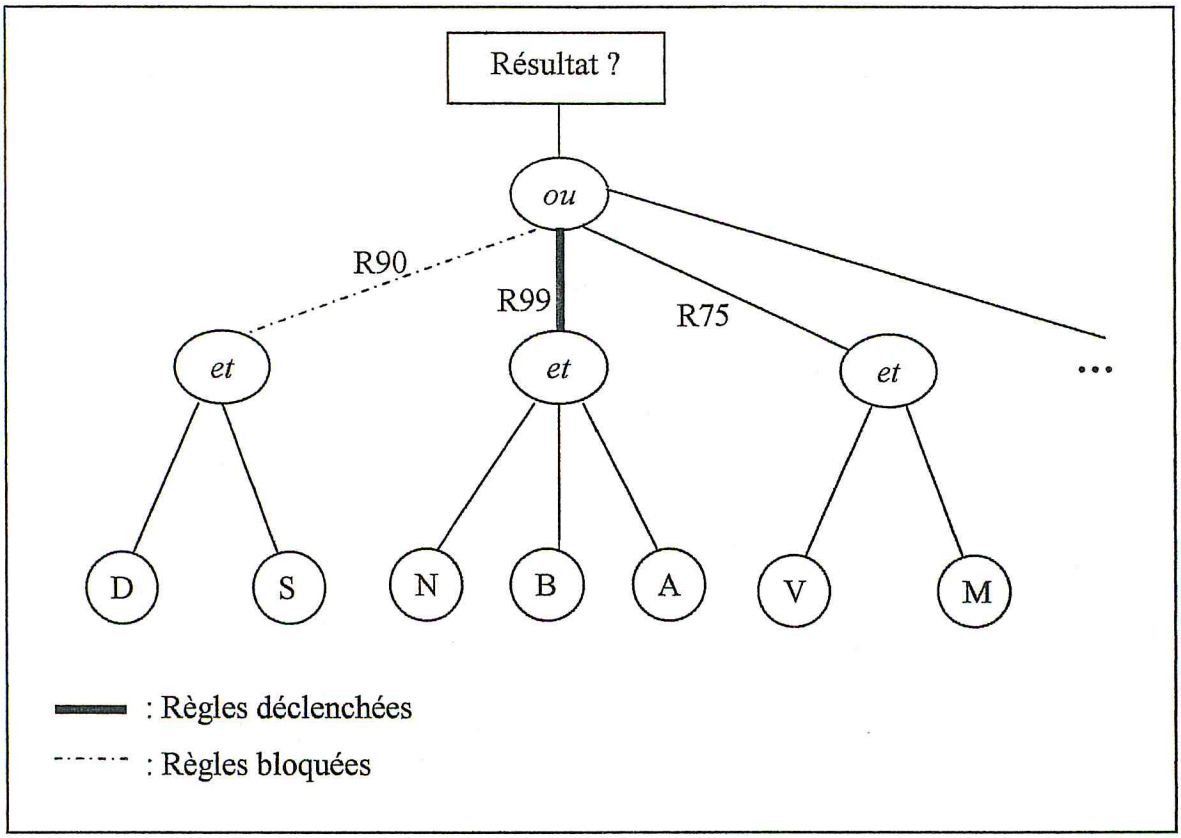


Figure III.12 - Partie de l'arborescence de déduction de Résultat

La figure suivante présente un dialogue où l'utilisateur désire savoir pourquoi le système n'a-t-il pas obtenu le résultat Résultat = Y :

Système : J'ai déduit le résultat Résultat = Z.

Utilisateur : Pourquoi pas Résultat = Y

Système : Il existe une règle qui conclut sur Résultat = Y, c'est la règle 'R90' :

[Si D
Et S
Alors Résultat = Y]

Or, cette règle a été bloquée (écartée) car au moins une de ses conditions n'est pas vérifiée (est invalide)

- D = faux (entré par l'utilisateur).

Figure III.13 - Exemple d'explication négative avec une seule règle écartée

Par contre si le nombre des règles N-candidates est supérieur à 1, le système présente une seule règle écartée et laisse les autres de coté comme étant des alternatives au cas où l'utilisateur n'est pas satisfait par l'explication donnée.

Exemple : Pour illustrer ce type d'explication, nous allons prendre les données suivantes :

Base de faits (BF) = {non A, non B, non R, M, S, Résultat = H},

Base de règle (BR) = {

R55 : *Si* A *Et* P *Alors* Résultat = C,
R61 : *Si* D *Et* B *Alors* Résultat = C,
R71 : *Si* R *Et* G *Alors* Résultat = C,
R74 : *Si* M *Et* S *Alors* Résultat = H
}

L'état de l'arbre ET/OU produit par le système est le suivant :

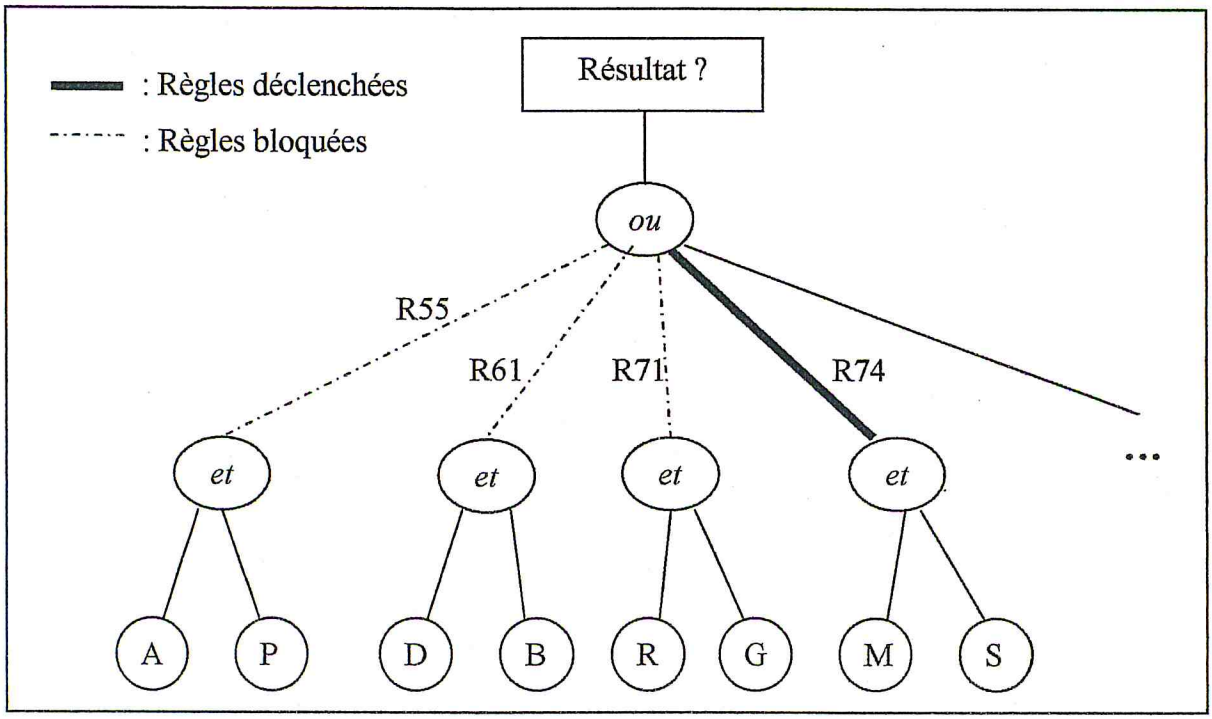


Figure III.14 - Partie de l'arborescence de déduction de Résultat = H

La figure suivante montre un dialogue où l'utilisateur demande des explications alternatives :

Système : J'ai déduit le résultat Résultat = H

Utilisateur : Pourquoi pas Résultat = C.

Système : Il existe une règle qui conclut sur Résultat = C, c'est la règle 'R55'.

[Si A
Et P
Alors Résultat = C]

Or, cette règle a été bloquée (écartée) car au moins une de ses conditions n'est pas vérifiée (invalide) :

- P = faux (entré par l'utilisateur).

Utilisateur : Autre explication.

Système : Il existe une règle qui conclut sur Résultat = C, c'est la règle 'R61'.

[Si D
Et B
Alors Résultat = C]

Or, cette règle a été bloquée (écartée) car au moins une de ses conditions n'est pas vérifiée (invalide) :

- B = faux (entré par l'utilisateur).

Utilisateur : Autre explication.

Système : Il existe une règle qui conclut sur Résultat = C, c'est la règle 'R71'.

[Si R
Et G
Alors Résultat = C]

Or, cette règle a été bloquée (écartée) car au moins une de ses conditions n'est pas vérifiée (invalide) :

- R = faux (entré par l'utilisateur).

Figure III.15 - Exemple d'explication négative avec plusieurs alternatives

Remarques :

- Nous avons envisagé dans le système SDPTV de limiter le nombre d'étapes à présenter à l'utilisateur lorsque ce nombre est trop important (supérieur à 5 par exemple).
- Une autre 'possibilité' implantée dans notre système résout le cas où l'on a plusieurs règles N-candidates, on se propose de les classer par nombre croissant d'étapes (niveaux). Ainsi on commence par présenter à l'utilisateur l'explication ayant le moins d'étapes.

ii) Règles en attente de déclenchement

Dans le deuxième cas, le module d'explication indique que la règle déclenchée est plus prioritaire que la ou les règles concluant sur le résultat R1 attendu par l'utilisateur.

Exemple : Reprenons l'arborescence de la figure III.12. Cette fois l'utilisateur s'attend au résultat : Résultat = T.

Système : J'ai déduit le résultat Résultat = Z.

Utilisateur : Pourquoi pas Résultat = T

Système : Il existe une règle qui conclut sur Résultat =T, c'est la règle 'R75' :

[Si V
Et M
Alors Résultat =T]

Or, cette règle n' pas été appliquée (déclenchée) car la règle déclenchée 'R99' est plus prioritaire.

Figure III.16 - Exemple d'explication négative avec une règle en attente de déclenchement

4.2. Explication selon le type de l'utilisateur

La fonction d'explication peut être utilisée par deux catégories d'utilisateurs dont les objectifs sont bien distincts:

- Le cogniticien (ingénieur de la connaissance)
- L'utilisateur final

4.2.1. L'ingénieur de la connaissance :

L'explication est très utile pour le cogniticien qui veut vérifier ou modifier la base de connaissances, elle lui permet de déboguer l'utilisation des connaissances par le système. De ce fait, l'explication attendue par le cogniticien doit faire abstraction de quelques détails qui lui sont évidents, donc inutiles, tel que les textes explicatifs des règles du système : il suffit juste de lui présenter le nom de la règle.

Exemple : Pour illustrer ce type d'explication, nous allons prendre les données suivantes :

Base de faits (BF) = {V, S, L, D, H, F},

Base de règle (BR) = {
R73 : *Si* S *Et* L *Alors* D,
R81 : *Si* ... *Alors* F,
R84 : *Si* V *Et* D *Et* H *Alors* F
}

L'état de l'arbre ET/OU produit par le système est le suivant :

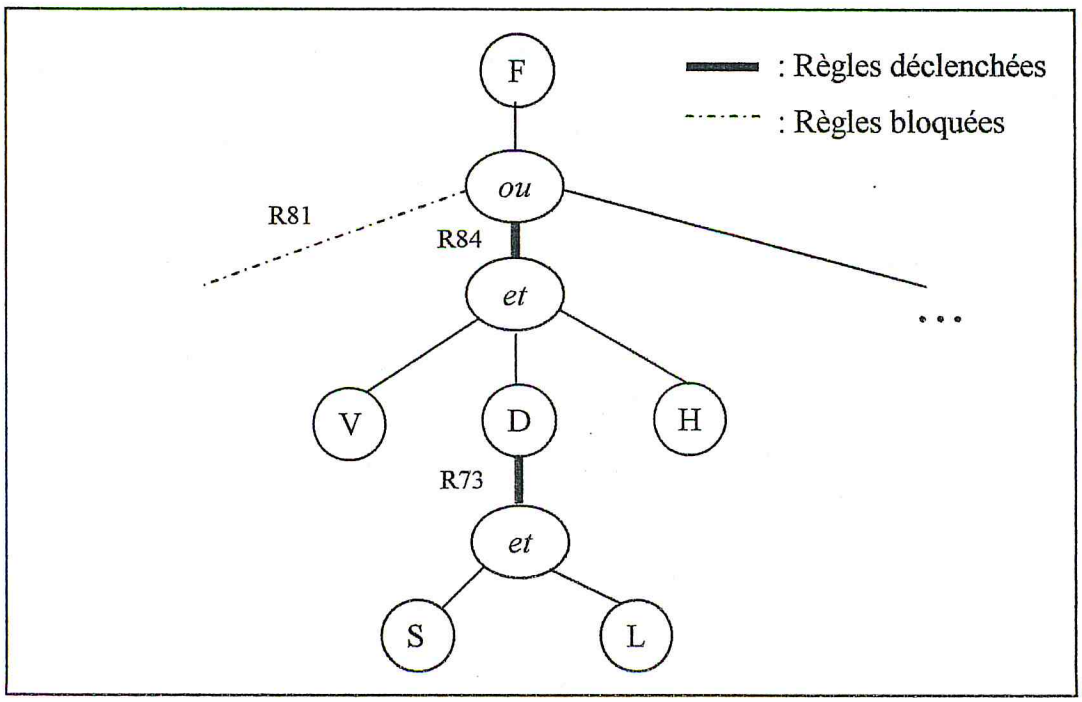


Figure III.17 - partie de l'arborescence de déduction du fait F

La figure suivante montre un dialogue où l'ingénieur de la connaissance demande des explications positives en fin de session :

Système : J'ai déduit le résultat Résultat = F.

Utilisateur : Comment avez-vous obtenu ce résultat ?

Système : Règle déclenchée : R84

Figure III.18 - Exemple d'explication destinée à l'ingénieur de la connaissance

4.2.2. L'utilisateur final :

Afin de satisfaire un grand nombre d'utilisateurs, le système est capable de fournir des explications selon le niveau de l'utilisateur. Néanmoins, ce dernier doit connaître son niveau pour l'indiquer au système en début de session.

Pour cela, les changements apportés au système se font essentiellement au niveau de la base de règles.

La figure suivante est un exemple d'une règle du système SEPT (Surveillance d'Equipements dans un Poste à très haute Tension) [Brézillon 87].

L'explication de niveau 1 est une explication minimale pour informer l'opérateur de l'existence d'un problème. L'explication de niveau 2 donne le nom de la protection de distance et l'état des trois phases. L'explication de niveau 3 donne la raison la plus usuelle alors qu'une autre raison est que la faute est sur une phase et la protection de distance s'est déclenchée de manière intempestive sur une phase. L'explication de niveau 4 correspond à la question classique "Pourquoi?" des autres systèmes experts. L'explication de niveau 5 fournit des connaissances stratégiques qui reflètent les choix au niveau de la compagnie qui veut que les perturbations du réseau entraînent le moins de désagrément possible pour les usagers.

SI l'ordre de déclenchement d'une protection de distance est émis sur deux phases,
ALORS *Action: Conclure à un fonctionnement anormal de la protection de distance;*
Explication de niveau 1: Envoyer un message d'avertissement;
Explication de niveau 2: Présentation des prémisses instanciées;
Explication de niveau 3: Une ouverture sur deux phases signifie probablement que la faute concerne trois phases et que la protection est défaillante sur une phase ;
Explication de niveau 4: L'ouverture d'un disjoncteur ne peut se faire que sur une ou trois phases ;
Explication de niveau 5: L'ouverture sur une phase préserve l'état courant du réseau alors qu'une ouverture sur trois phases isole totalement l'élément avec la faute.

Figure III.19 - Exemple de règle avec plusieurs niveaux d'explication

Le système SDPTV propose deux niveaux d'explication :
- Débutant (novice),
- Expert (Avancé).

Le niveau expert est un cas particulier correspondant à un expert dans le domaine considéré, dans notre cas il s'agit d'un expert en maintenance de télévisions.

Pour un expert, nous avons jugé inutile de le surcharger avec les différents détails du raisonnement (diagnostic). Le système se contente ainsi de présenter les grandes lignes du raisonnement.

Exemple 1 : Pour illustrer l'explication positive destinée à un expert, nous allons prendre les données suivantes :

Base de faits (BF) = {non L, non M, F, A, J, K, H, D, E, V, S, Résultat = P},

Base de règle (BR) = {

R11 : *Si* L *Alors* N,
R13 : *Si* A *Et* J *Et* K *Alors* H
R14 : *Si* M *Alors* N,
R21 : *Si* V *Alors* S,
R22 : *Si* Y *Alors* S,
R27 : *Si* F *Et* H *Alors* D,
R33 : *Si* N *Et* B *Alors* Résultat = O,
R39 : *Si* D *Et* E *Et S Alors* Résultat = P
}

L'état de l'arbre ET/OU produit par le système est le suivant :

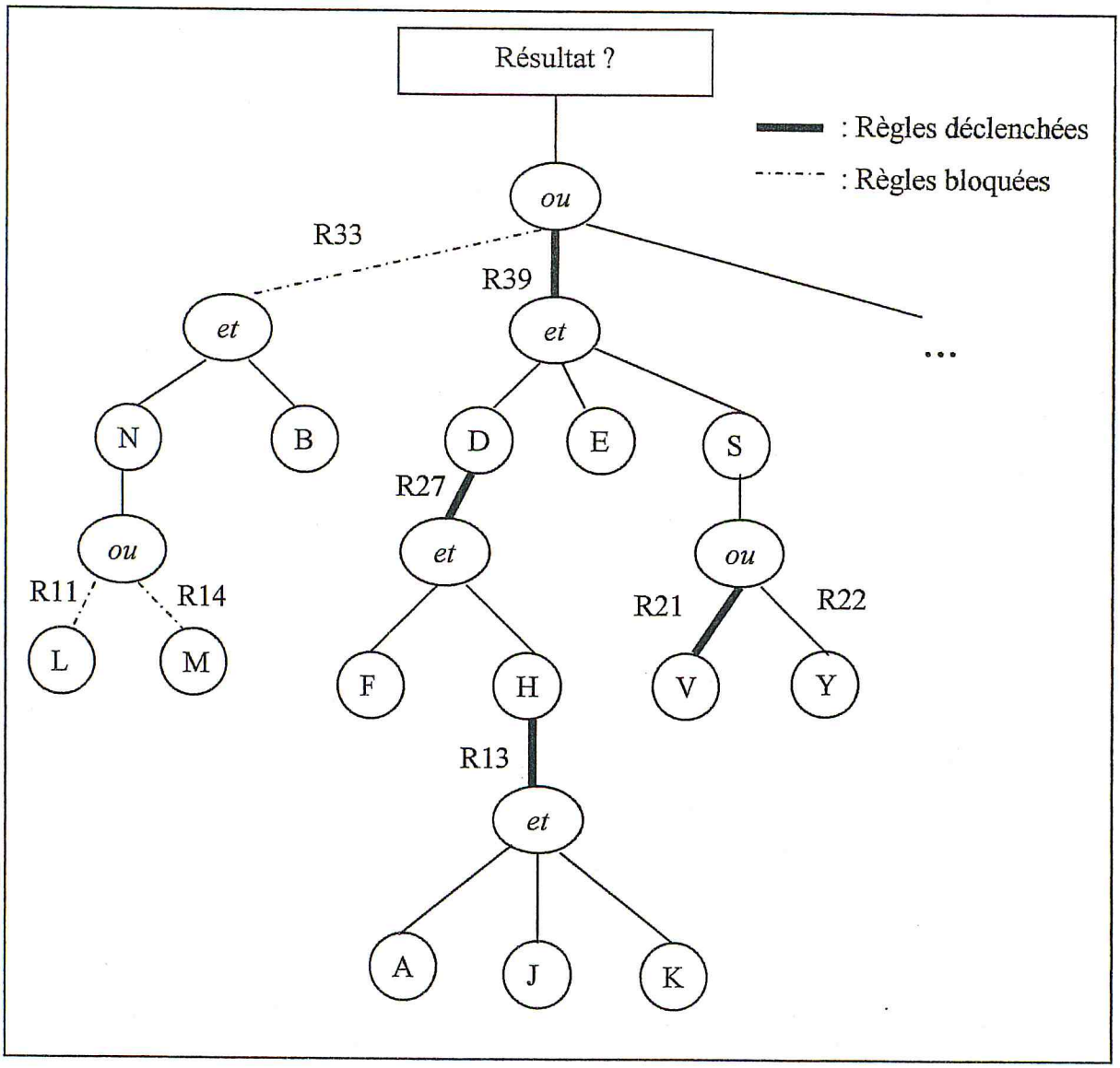


Figure III.20 - Partie de l'arborescence de déduction du fait Résultat

La figure suivante montre un dialogue où l'expert demande des explications positives en fin de session :

Système : J'ai déduit le résultat : Résultat = P.

Utilisateur : Comment as-tu obtenu ce résultat ?

Système : Je sais que :

- F
- A
- J
- K
- E
- Y

J'ai donc déclenché respectivement les règles suivantes :

- R13
- R27
- R21
- R39

Ce qui m'a permis de conclure sur P.

Figure III.21 - Exemple d'explication positive destinée à un expert

Exemple 2 : Reprenons l'arborescence de déduction de la figure III.20.

Système : J'ai déduit le résultat : Résultat = P.

Utilisateur : Pourquoi pas Résultat = O.

Système : Pour conclure sur Résultat = O, il faut que :

- L soit vrai.

Or, vous avez entré que L est faux.

Utilisateur : Autre explication.

Système : Pour conclure sur résultat = O, il faut que :

- M soit vrai.

Or, vous avez entré que M est faux.

Figure III.22 - Exemple d'explication négative destinée à un expert

Exemple 3 : Pour illustrer l'explication positive destinée à un expert, nous allons prendre les données suivantes :

Base de faits (BF) = {H, non N, C},

Base de règle (BR) = {

R9 : *Si* H *Et* B *Alors* Résultat = U,
R13 : *Si* P *Et* M *Alors* Résultat = W,
R15 : *Si* S *Et* R *Et F Alors* Résultat = Z,
R23 : *Si* N *Alors* B,
R29 : *Si* J *Alors* B,
R32 : *Si* K *Alors* B,
R36 : *Si* C *Et* D *Alors* J

}

L'état de l'arbre ET/OU produit par le système est le suivant :

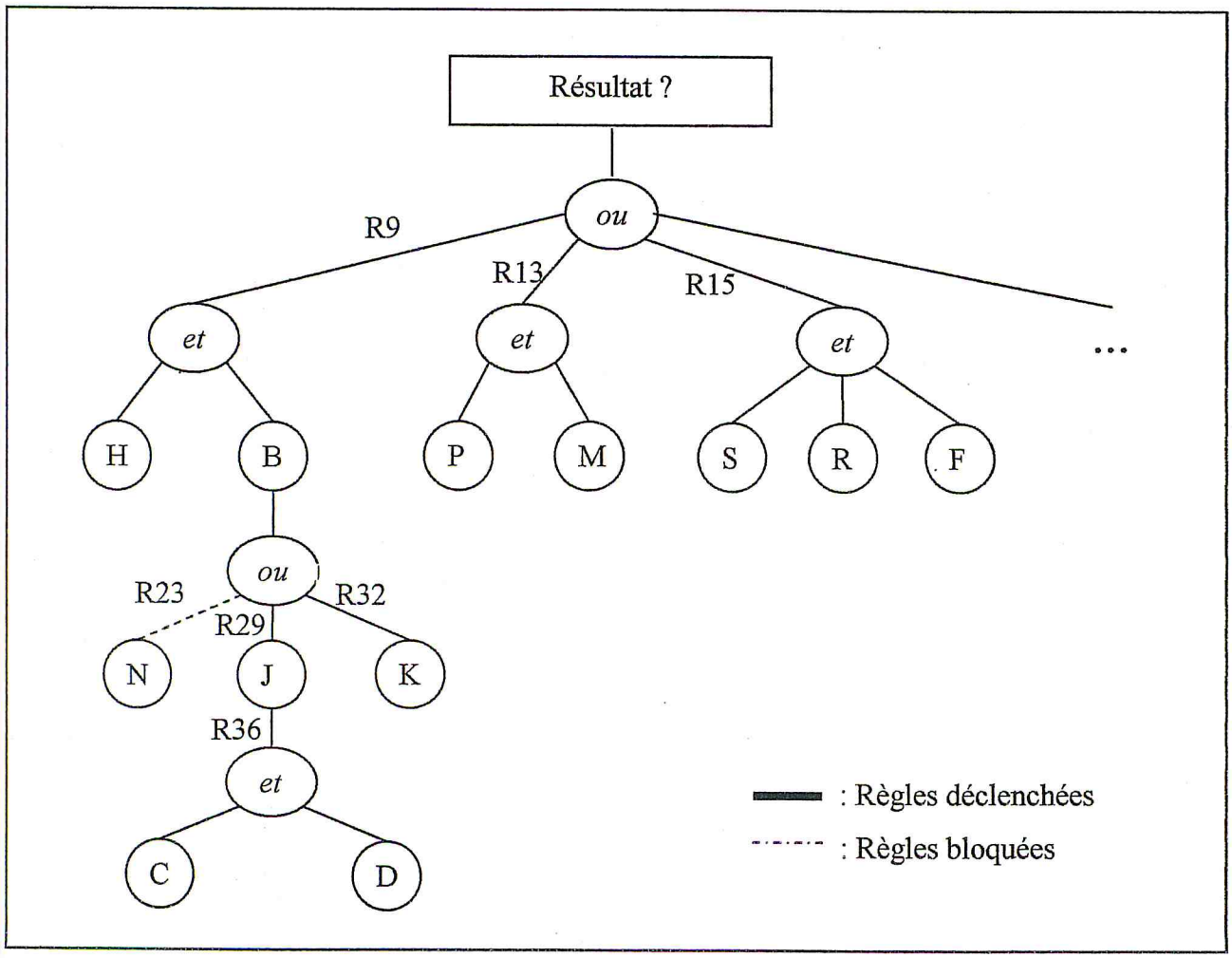


Figure III.23 - Partie de l'arborescence de déduction du fait Résultat

La figure suivante montre un dialogue où l'expert demande des justifications sur la question qui lui a été posée :

Système : Quelle est la valeur de D ?

Utilisateur : Pourquoi me posez vous cette question ?

Système : Je cherche à conclure sur Résultat = U.

Pour cela, j'ai besoin de connaître la valeur de D.

Figure III.24 – Exemple d'explication (justification d'une question) destinée à l'expert

5. Description de la réalisation

Pour la réalisation du système expert explicatif, nous avons choisi comme domaine d'application le diagnostic de pannes de TéléVision, car d'une part c'est un domaine familier et d'autre part il utilise des connaissances très techniques qui nécessitent une Explication.

5.1. Objectifs du système SDPTV

Les objectifs du système SDPTV (Système de Diagnostic de Pannes de TéléVision) sont nombreux :

- Aider ou assister un technicien en maintenance de TéléVisions à détecter les défauts de fonctionnement et à localiser les différentes pannes.
- Offrir une facilité d'explication afin de justifier le raisonnement poursuivi et de convaincre l'utilisateur de la validité des résultats obtenus.

5.2. Caractéristiques du système SDPTV

SDPTV est un système à base de connaissances dont les spécifications sont les suivantes :

- Le moteur du système utilise la logique d'ordre 0+, ce qui permet d'introduire les opérateurs : = , <> , < , <= , > , >=.

Exemple :

Tension_Alimentation_Secondaire >9

- Le moteur fonctionne en chaînage mixte. Il offre ainsi une grande interactivité grâce aux questions qu'il pose à l'utilisateur et qui permettent de diriger le raisonnement.
- La mise en œuvre de la base de règles s'est faite en collaboration avec un expert en maintenance de télévisions. Après plusieurs entrevues avec celui-ci une base de 50 règles a été élaborée (voir Annexe).
- Le système SDPTV permet de détecter une trentaine (30) de pannes différentes.

- Les pannes détectées sont classées en six (06) catégories :
  - Pannes totales,
  - Ecran noir,
  - Défauts de géométrie,
  - Couleur de l'image,
  - Pannes son (audio)
  - Pannes diverses (télécommande, bus I2C, etc...)
- Une caractéristique essentielle du système SDPTV qui le distingue des autres systèmes de diagnostic est sa capacité à expliquer son raisonnement en des termes qu'un utilisateur du système peut facilement comprendre. Il ne se contente pas ainsi de présenter la panne détectée à l'utilisateur mais va plus loin en convaincant ce dernier de la justesse et de la validité de ces solutions. Cette caractéristique permet d'augmenter considérablement la crédibilité du système SDPTV envers ses utilisateurs.

5.3. Implémentation de la base de connaissances

La base de connaissances du système est stockée sous forme d'un fichier XML (eXtensible Markup Langage). Le XML est un langage à base de balises permettant de créer et de manipuler des documents structurés. L'utilisation d'un tel langage présente plusieurs avantages parmi lesquels la possibilité de définir des *schémas* (appelés aussi XSD : XML Schema Definition) qui définissent la structure du document. Ces schémas sont d'une importance capitale : ils permettent de définir la syntaxe du langage (grammaire formelle) ainsi que sa sémantique.

La figure suivante montre un exemple de déclaration d'une règle du système SDPTV :

```
<Regle nom="regle001" priorite="5">
        <Si>
                <Condition>
                        <Nom-attribut>Prise_Secteur_Raccordée</Nom-attribut>
                        <Comparateur>=</Comparateur>
                        <Valeur>Non</Valeur>
                </Condition>
        </Si>
        <Alors>
                <Conclusion>
                        <Nom-attribut>Panne</Nom-attribut>
                        <Valeur>PRISE SECTEUR Non raccordée</Valeur>
                </Conclusion>
        </Alors>
        <Explication>Si la prise secteur n'est pas raccordée Alors le TéléViseur ne peut démarrer</Explication>
</Regle>
```

Figure III.25 - Exemple d'une règle

La figure suivante montre un exemple de déclaration d'un attribut :

```
<Attribut>
        <Nom>Mémorisation_chaînes</Nom>
        <Type>chaine</Type>
        <Question>La mémorisation des chaines fonctionne-t-elle ?</Question>
</Attribut>
```

Figure III.26 - Exemple d'un attribut

La figure suivante montre un exemple de déclaration d'un fait :

```
<Fait>
        <Nom-attribut>Réception_se_bloque_sur_une_chaîne</Nom-attribut>
        <Valeur>Oui</Valeur>
</Fait>
```

Figure III.27 - Exemple d'un fait

La figure suivante montre une partie du schéma XML qui concerne la déclaration d'une règle :

```
<xsd:element name="Regle">
        <xsd:complexType>
                <xsd:sequence>
                        <xsd:element ref="Si"/>
                        <xsd:element ref="Alors"/>
                        <xsd:element name="Explication" type="xsd:string" minOccurs="0"/>
                </xsd:sequence>
                <xsd:attribute name="nom" use="required"/>
                <xsd:attribute name="priorite" default="5"/>
        </xsd:complexType>
</xsd:element>
```

Figure III.28 - Partie du schéma XML concernant la représentation d'une règle

5.4. Différentes classes du système

Ayant utilisé le langage JAVA qui est un langage orienté objet pour le développement du système, l'ensemble du système est représenté en classes.

La figure ci-dessous met en relief les différentes classes du système, à savoir :

- la classe GUI(Graphical User Interface) représentant l'interface graphique du système.
- la classe MEXP représentant le module d'explication
- la classe MI représentant le moteur d'inférences
- la classe MAC représentant le module d'acquisition des connaissances
- la classe MVDC représentant le module de validation des connaissances
- la classe KB (Knowledge Base) représentant la base de connaissances

- la classe BR représentant la base de règles
- la classe BF représentant la base de faits
- la classe DDF représentant le dictionnaire de faits
- la classe REGLE représentant objets de type règle
- la classe FAIT représentant les objets de type fait
- la classe ATTRIBUT représentant les objets de type attribut

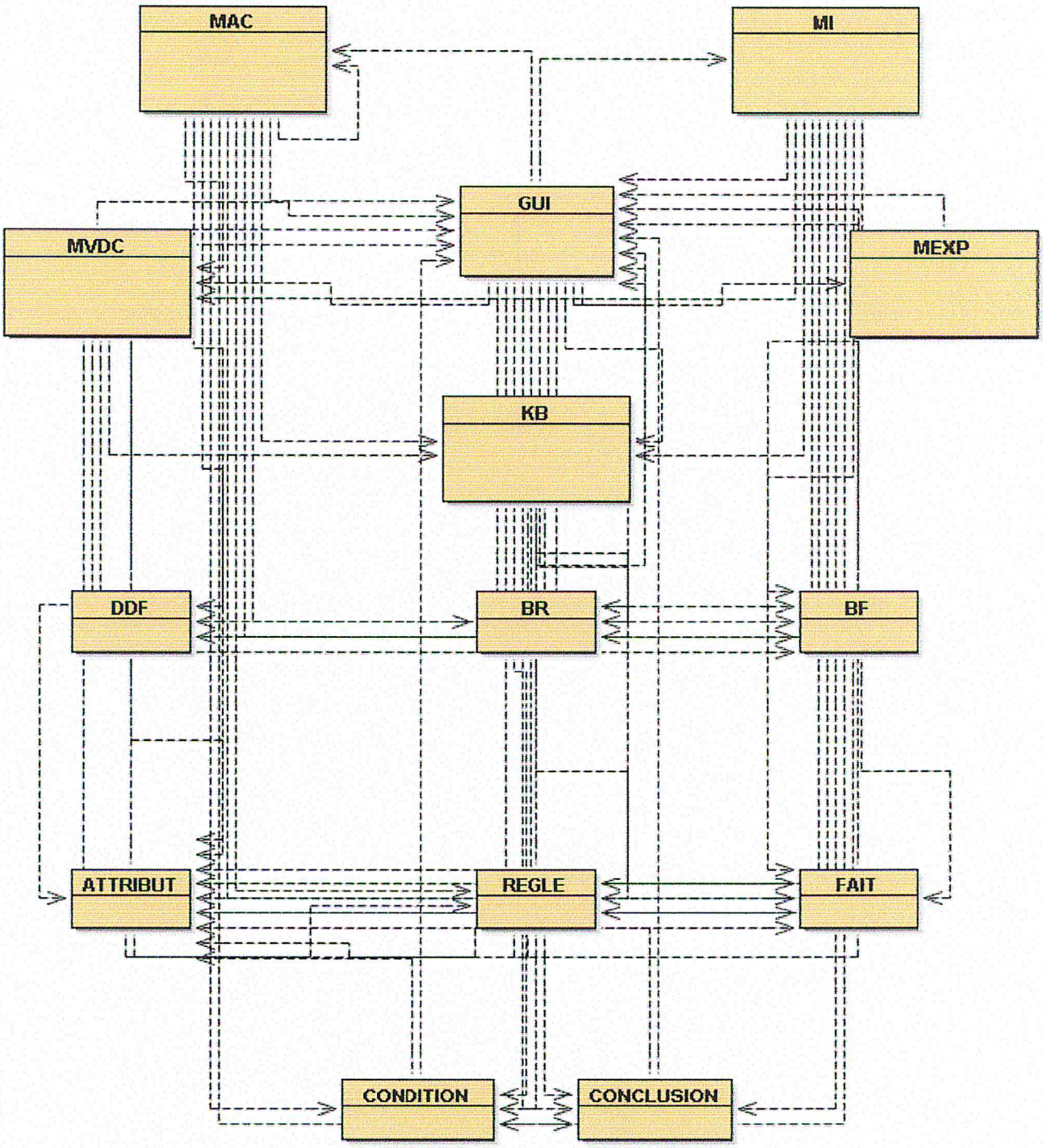


Figure III.29 - Différentes classes du système SDPTV

Remarque : l'image de la figure III.29 a été réalisée avec l'éditeur JAVA BlueJ.

5.4.1. La classe KB (Base de Connaissances)

Cette classe représente la base de connaissances du système.

Ses attributs sont :
- baseFait : la base de faits du système
- baseRegle : la base de règles du système
- dicFait : le dictionnaire des faits du système

Ses méthodes sont :
- getBf : retourne la base de faits du système
- getBr : retourne la base de règles du système
- getDdf : retourne le dictionnaire de faits du système
- ReInit : réinitialise la base de connaissances (réinitialisation interne)

5.4.2. La classe GUI (Interface Graphique)

Cette classe est responsable du lancement du système. Elle permet d'une part la communication entre les différents modules et d'autre part la communication entre l'utilisateur et le système.

Ses attributs sont :
- moduleExp : variable globale représentant le module d'explication.
- moduleVal : variable globale représentant le module de validation de connaisances.
- moduleAcq : variable globale représentant le module d'acquisition de connaissances.
- baseCon : variable globale représentant la base de connaissances.
- moteurInf : variable globale représentant le moteur d'inférences du système.
- barreMenu : barre de menu de l'interface graphique
- btnCom : bouton commencer, il invoque la méthode Commencer.

Ses méthodes sont :
- Commencer : permet de lancer une nouvelle session de diagnostic.
- PoserQuestion : permet l'affichage des questions posées à l'utilisateur.

5.4.3. La classe MI (Moteur d'Inférences)

Ce module est la partie dynamique du système : c'est son noyau.

Ses attributs sont :
- listeRgNonBloq : la liste des règles non bloquées (de type Vector)
- continuer : est à FAUX si l'utilisateur interrompt la session (de type booléen).
- traceRais : représente la trace de raisonnement du système.

Ses méthodes sont :
- ChainageInterrompu : retourne VRAI si le processus de résolution a été interrompu par l'utilisateur.

- ArreterChainage : met à VRAI la variable « continuer ».
- ReInit : réinitialise la liste des règles non bloquées.
- EvaluerCondition : elle prend en paramètre une variable de type CONDITION et essaye de l'évaluer selon le contenu de la base de faits.
- FiltrageAvant : effectue l'opération de filtrage pour le chaînage avant.
- FiltrageArrière : effectue l'opération de filtrage pour le chaînage arrière.
- ResolutionConflits : effectue l'opération de résolution de conflits.
- Application : effectue l'opération de déclenchement d'une règle.
- ChainageAvant : déclenche le chaînage avant.
- ChainageArriere : déclenche le chaînage arrière.
- ChainageMixte : déclenche le chaînage mixte.

5.4.4. La classe MVDC (Module de Validation de Connaissances)

Ce module est responsable de la cohérence de la base de connaissances.

Ses méthodes sont :
- VerifierKb : elle permet de vérifier que le fait passé en paramètre n'est pas en contradiction avec la base de faits.

5.4.5. La classe MAC (Module d'Acquisition de Connaissances)

Ce module a deux fonctions principales. La première est celle de l'initialisation de la base de connaissances (lecture du fichier XML « SDPTV.XML »). La deuxième est de faciliter la modification de la base de connaissances.

Ses attributs sont :
- builder : c'est un objet permettant de créer un arbre à partir d'un document XML (utilisé lors de l'initialisation de KB)
- outputter : c'est un objet permettant la création d'un document XML à partir d'un arbre (utilisé lors de l'enregistrement des modifications apportées à KB).
- doc : représente l'arbre du document XML (généré par l'objet « builder »).

Ses méthodes sont :
- InitKb : initialise la base de connaissances.
- Afficher : affiche l'interface graphique du MAC.
- AjouterRegle, ModifierRegle, SupprimerRegle : pour la modification de la base de règles.
- AjouterFait, ModifierFait, SupprimerFait : pour la modification de la base de faits.
- AjouterAttribut, ModifierAttribut, SupprimerAttribut : pour la modification du dictionnaire de faits.

5.4.6. La classe MEXP (Module d'EXPlication)

Ce module est chargé de la production et de la présentation d'une explication à l'utilisateur.

Ses attributs sont :
- traceRais : la trace du raisonnement contenant toutes les informations dont a besoin ce module.

Ses méthodes sont :
- Afficher : affiche l'interface graphique du module d'explication.
- ExplicationQuestion : construit une explication du type pourquoi me poses-tu cette question.
- ExpliquerComment : construit une explication du type comment a-t-on obtenu tel fait ou tel résultat.
- ExpliquerPourquoiPas : construit une explication du type pourquoi n'as-tu pas établi tel diagnostic.

5.4.7. La classe BF (Base de Faits)

Cette classe représente la base de faits : c'est la mémoire de travail du système. Elle s'occupe de la gestion des faits.

Ses attributs sont :
- nbF : nombre de faits dans la base.
- listeF : la liste des faits (de type Vector[]).

Ses méthodes sont :
- AjouterFait : ajoute un fait à la base de faits.
- getFait : retourne la référence du fait dont le nom est passé en paramètre, nul sinon.
- getListeFait : retourne la liste des faits.
- getNbFait : retourne le nombre de faits dans la base.
- ReInit : réinitialise la base de faits.

5.4.8. La classe BR (Base de Règles)

Cette classe représente la base de règles du système. Elle s'occupe de la gestion des règles.

Ses attributs sont :
- nbR : nombre de règles dans la base.
- listeR : la liste des règles (de type Vector).

Ses méthodes sont :
- AjouterRegle : ajoute une règle à la base.
- getListeR : retourne la liste des règles.
- ReInit : réinitialise la base de règles.

5.4.9. La classe DDF (Dictionnaire Des Faits)

Cette classe représente le dictionnaire des faits du système. Elle s'occupe de la gestion des Attributs.

Ses attributs sont :
- nbAtt : nombre d'Attributs dans la base.
- listeAtt : la liste des Attributs (de type Vector[]).

Ses méthodes sont :

- AjouterAtt : ajoute un Attribut au dictionnaire.
- getListeAtt : retourne la liste des Attributs
- getAtt : retourne la référence de l'Attribut dont le nom est passé en paramètre, nul sinon.
- ReInit : réinitialise le dictionnaire des faits.

5.4.10. La classe FAIT

Les faits constituent les éléments de base de la classe BF. Ils représentant aussi les éléments clés du moteur d'inférences.

Ses attributs sont :
- attCor : référence de l'attribut correspondant.
- val : valeur du fait.

Ses méthodes sont :
- getAttCor : retourne la référence de l'attribut correspondant.
- getVal : retourne la valeur du fait.

5.4.11. La classe ATTRIBUT

Les Attributs constituent les éléments de base de la classe DDF.

Ses attributs sont :
- nom : nom de l'Attribut.
- type : type de l'Attribut (booléen, numérique ou chaîne de caractères).
- listeValAdmis : liste des valeurs admises pour cet Attribut (elle est à nul s'il s'agit par exemple d'un Attribut numérique).
- Question : question sur l'Attribut (s'il est demandable).
- listeRgCond : liste des règles où cet Attribut est en condition.
- listeRgConc : liste des règles où cet Attribut est en conclusion.
- FaitCor : fait correspondant à l'Attribut dans la base de faits.

Ses méthodes sont :
- getNom : retourne le nom de l'Attribut.
- getType : retourne le type de l'Attribut.
- getListeRgCond : retourne la liste des règles où cet Attribut est en condition.
- getListeRgConc : retourne la liste des règles où cet Attribut est en conclusion.
- setFaitCor : met à jour la variable FaitCor.
- getFaitCor : retourne le fait correspondant à l'Attribut dans la base de faits.
- AjouterRgCondition : ajoute la référence d'une règle à la liste listeRgCond.
- AjouterRgConclusion : ajoute la référence d'une règle à la liste listeRgConc.
- getListeValAdmis : retourne la liste des valeurs admises.
- getQuestion : retourne la question associée à cet Attribut.
- ReInit : réinitialise l'Attribut.

5.4.12. La classe REGLE

Les Règles constituent les éléments de base de la classe BR.

Ses attributs sont :
- nom : nom de la règle.
- prio : priorité de la règle.
- nbCdt : nombre de conditions de la règle.
- nbCnc : nombre de conclusions de la règle.
- nbCdtNonVerif : nombre de conditions non vérifiées.
- bloq : indique l'état de la règle : VRAI indique que la règle est bloquée.
- listeCond : liste des conditions de la règle.
- listeConc : liste des conclusions de la règle.
- explication : explication de la règle.

Ses méthodes sont :
- getNom : retourne le nom de la règle.
- getPrio : retourne la priorité de la règle.
- setPrio : met à jour la priorité de la règle.
- getNbCdt : retourne le nombre de conditions.
- getCond : retourne la liste des conditions.
- AjouterCond : ajoute une condition à la liste des conditions.
- getNbCnc : retourne le nombre de conclusions.
- getConc : retourne la liste des conclusions.
- AjouterConc : ajoute une conclusion à la liste des conclusions.
- ValiderCond : décrémente le nombre des conditions non vérifiées.
- EstDeclenchable : retourne VRAI si le nombre de conditions non vérifiées est nul.
- setBloquee : met a VRAI la variable bloq.
- estBloquee : retourne VRAI si la règle est bloquée.
- setExplication : met à jour l'explication de la règle.
- getExplication : retourne l'explication de la règle.
- ReInit : réinitialise la règle.

5.4.13. La classe CONDITION

Les conditions représentent la partie SI d'une règle.

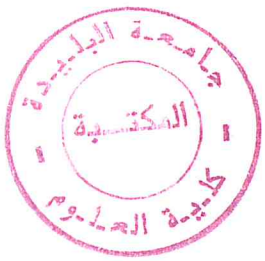
Ses attributs sont :
- operande1 : premier opérande.
- comparateur : un des comparateurs : =, #, <, <=, >, >= .
- operande2 : deuxième opérande.
- typeComp : type de la comparaison (0)pour attribut-attribut, (1)pour attribut-valeur
- etat : indique l'état de la condition.

Ses méthodes sont :
- getOp1 : retourne le premier opérande.
- getOp2 : retourne le deuxième opérande.
- getComp : retourne le comparateur.
- getTypeComp : retourne le type de comparaison.
- getEtat : retourne l'état de la règle.
- setEtat : met à jour l'état de la règle.
- ReInit : réinitialise la règle.

5.4.14. La classe CONCLUSION

Les conclusions représentent la partie ALORS d'une règle.
Ses attributs sont :
- operande1 : premier opérande.
- operande2 : deuxième opérande.
- typeAffect : type d'affectation (0)pour attribut-attribut, (1)pour attribut-valeur.

Ses méthodes sont :
- getOp1 : retourne le premier opérande.
- getOp2 : retourne le deuxième opérande.
- getTypeAffect : retourne le type d'affectation.



5.5. L'interface

L'interface graphique permet de communiquer avec le système de manière naturelle. Toutes les commandes possibles sont rangées dans des boutons ou des menus. La figure ci-dessous représente l'interface du système SDPTV :

Figure III.30 - Interface graphique du système SDPTV

Pour commencer une nouvelle session il suffit de cliquer sur le bouton : Session -> Nouvelle du menu principal. L'utilisateur pourra alors choisir son niveau si c'est un utilisateur final ou d'entrer un mot de passe s'il s'agit de l'ingénieur de la connaissance. La figure ci-dessous illustre cette opération :

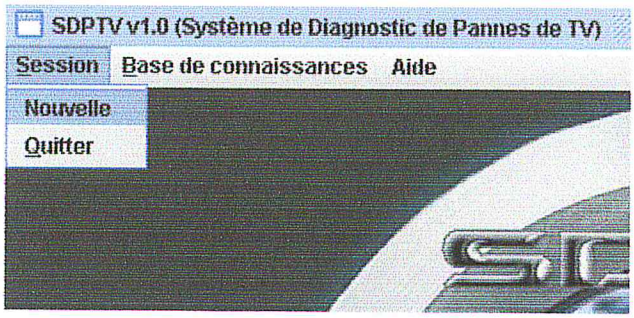


Figure III.31 - Bouton de création d'une nouvelle session

L'interface permet aussi la visualisation du fichier de la base de connaissances. Il suffit de cliquer sur : Base de connaissances -> Voir. La figure ci-dessous illustre cette opération :

Figure III.32 - Bouton de visualisation du fichier de la base de connaissances

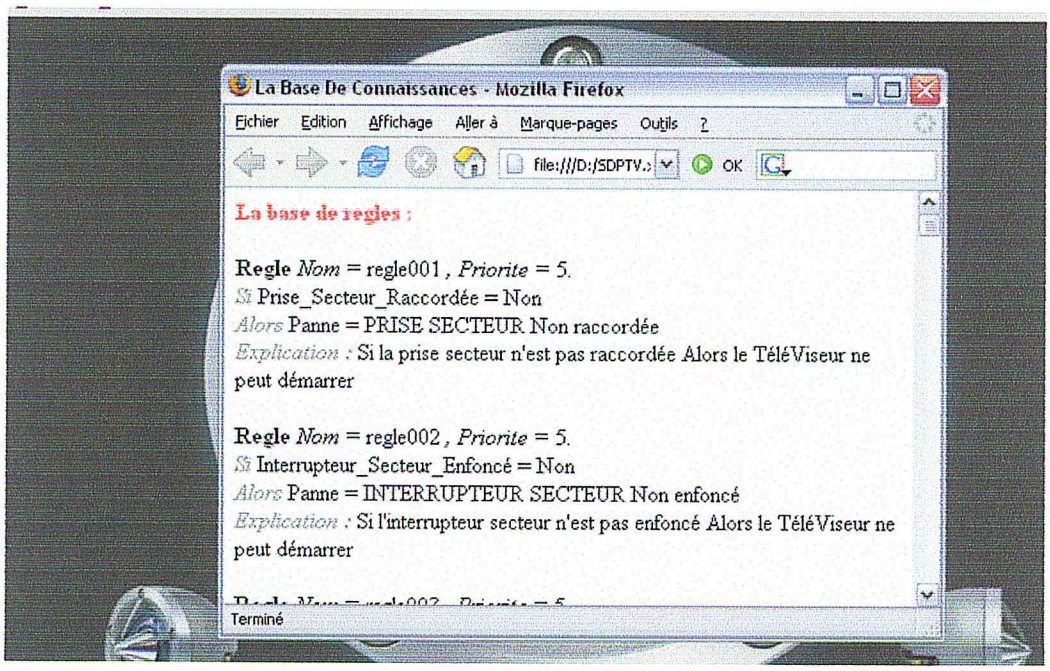


Figure III.33 – Visualisation du fichier de la base de connaissances

L'interface permet aussi la modification du fichier de la base de connaissances (ceci est rendu possible grâce au module d'acquisition des connaissances). Il faudra toutefois s'identifier comme étant l'ingénieur de la connaissance pour avoir les privilèges nécessaires. La figure ci-dessous présente le module d'acquisition des connaissances :

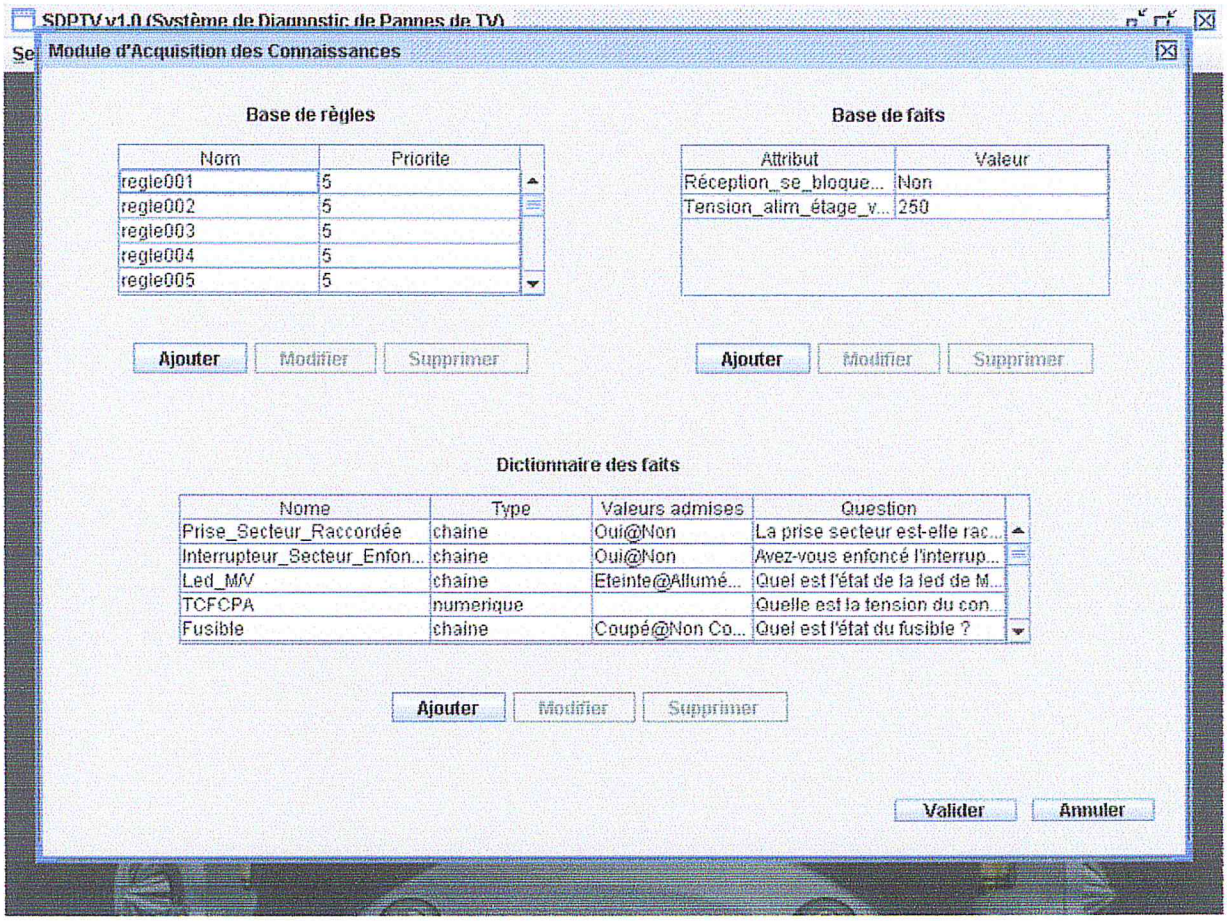


Figure III.34 – Module d'acquisition des connaissances

Il est ainsi possible de modifier un fait, une règle ou un attribut. Les deux figures suivantes illustrent respectivement la modification d'un fait de type chaîne et de type numérique :

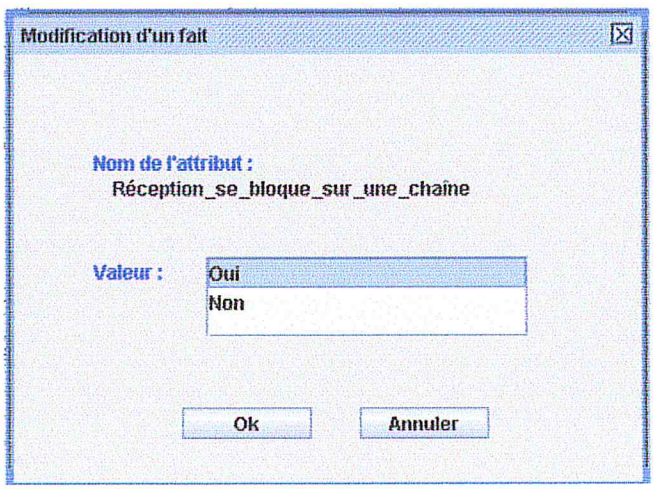


Figure III.35 – Modification d'un fait de type chaîne

Figure III.36 – Modification d'un fait de type numérique

5.6. Le moteur d'inférences

Au niveau graphique ce module est chargé de poser des questions à l'utilisateur ou de l'informer des résultats intermédiaires. Les deux figures suivantes représentent des questions posées à l'utilisateur :

Figure III.37 – Exemple de question posée à l'utilisateur

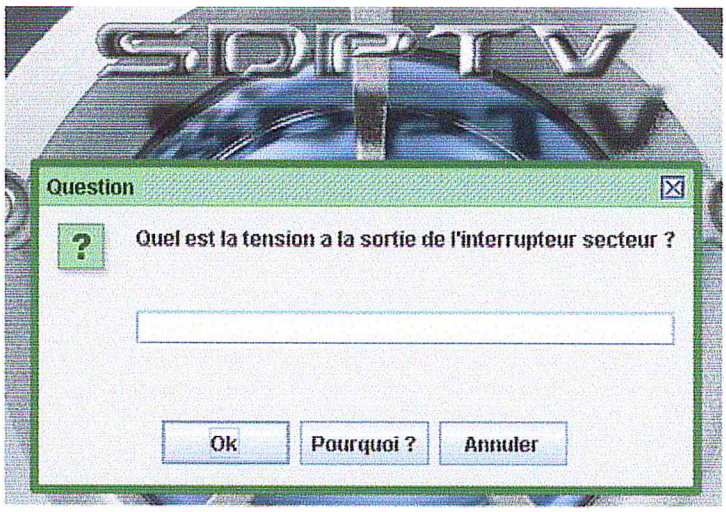


Figure III.38 – Exemple de question posée à l'utilisateur

En fin de session le système affiche le diagnostic établi. La figure suivante illustre ceci :

Figure III.39 – Message de fin de session

5.7. Le module d'explication

Ce module est chargé de la construction et de la transmission d'une explication suivant le niveau de l'utilisateur et le type d'explication demandée.

La figure suivante est un exemple de justification d'une question posée à un utilisateur débutant :

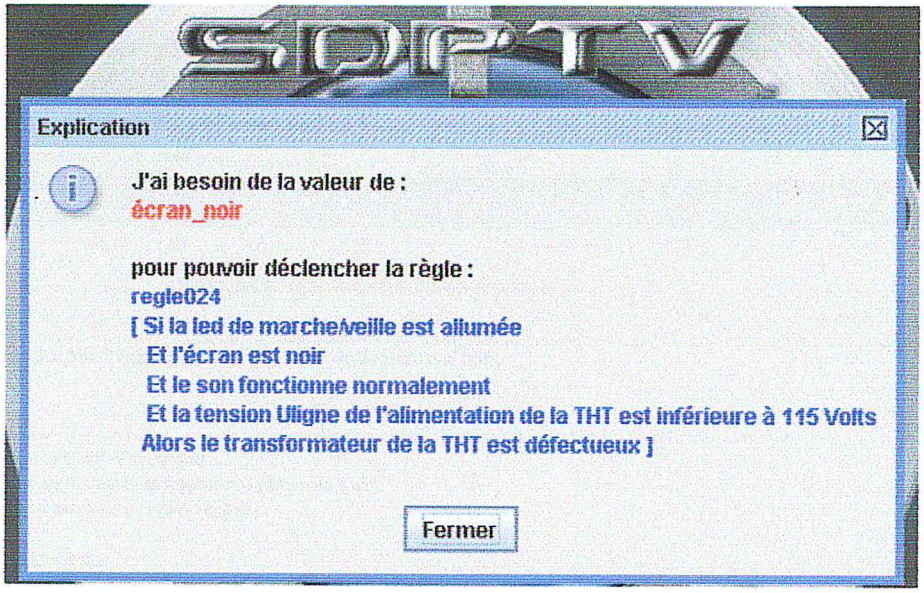


Figure III.40 – Exemple de justification d'une question posée à un utilisateur débutant

La figure suivante est un exemple d'explication positive en fin de session destinée à un utilisateur débutant :

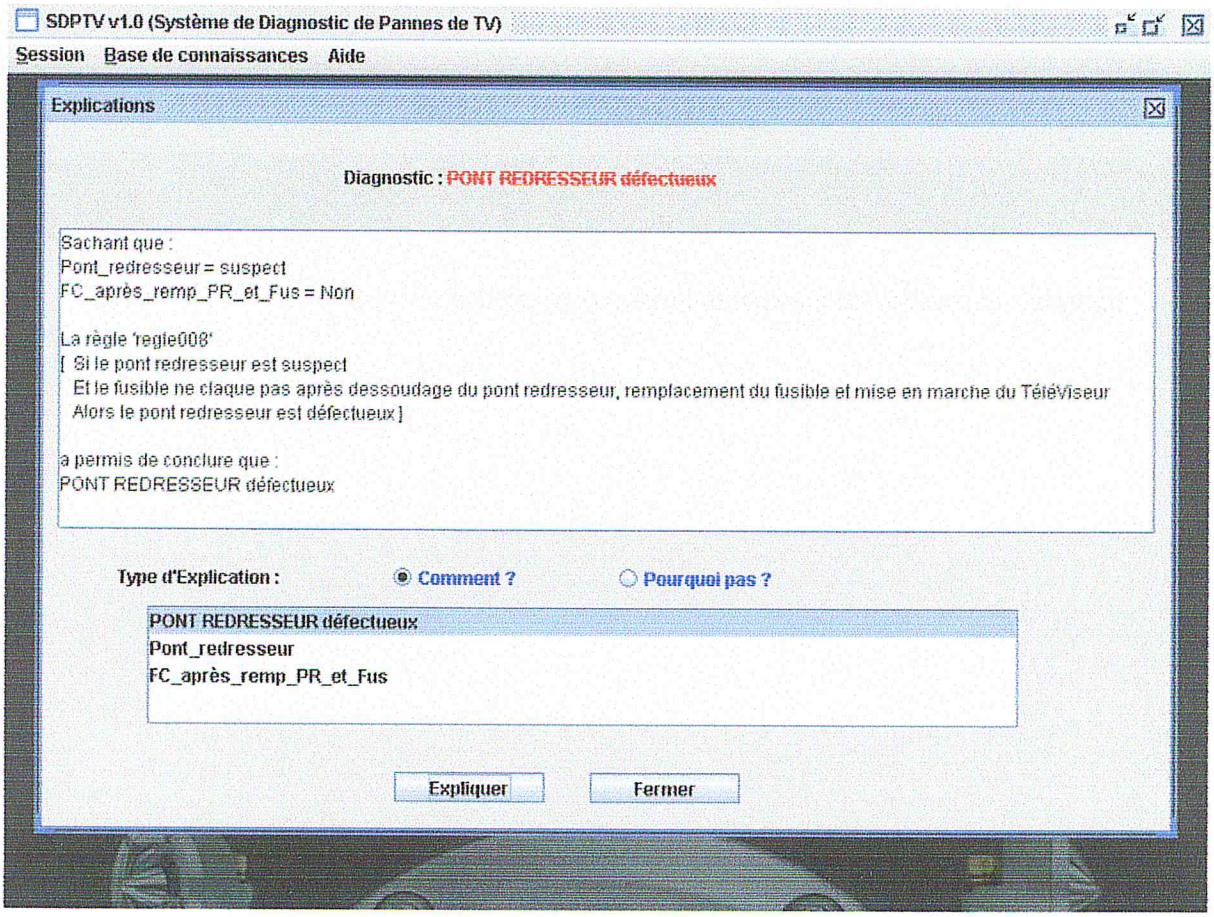


Figure III.41 – Exemple d'explication positive destinée à un utilisateur débutant

6. Conclusion

Dans ce chapitre, nous avons présenté en détail le système SDPTV. Nous avons décrit ses différentes composantes :

- la base de connaissances,
- le moteur d'inférences,
- le module de validation des connaissances,
- le module d'explication,
- l'interface.

Lors de la conception du module d'explication, nous avons présenté les différents types d'explications que propose le système :

- Explications en cours de session
- Explication en cours de session

Nous constatons que le système est capable d'adapter ses explications à deux catégories d'utilisateurs : à l'ingénieur de la connaissance et à l'utilisateur final.

L'implémentation de la base de connaissances s'est faite en langage XML qui présente plusieurs avantages.
Quant à l'implémentation du système, elle a été réalisée en langage JAVA, ce qui permet d'organiser le système en classes.

Enfin, nous avons donné quelques exemples de sessions avec le système SDPTV afin d'illustrer la maquette réalisée.

La figure suivante est un exemple d'explication négative en fin de session destinée à un utilisateur débutant :

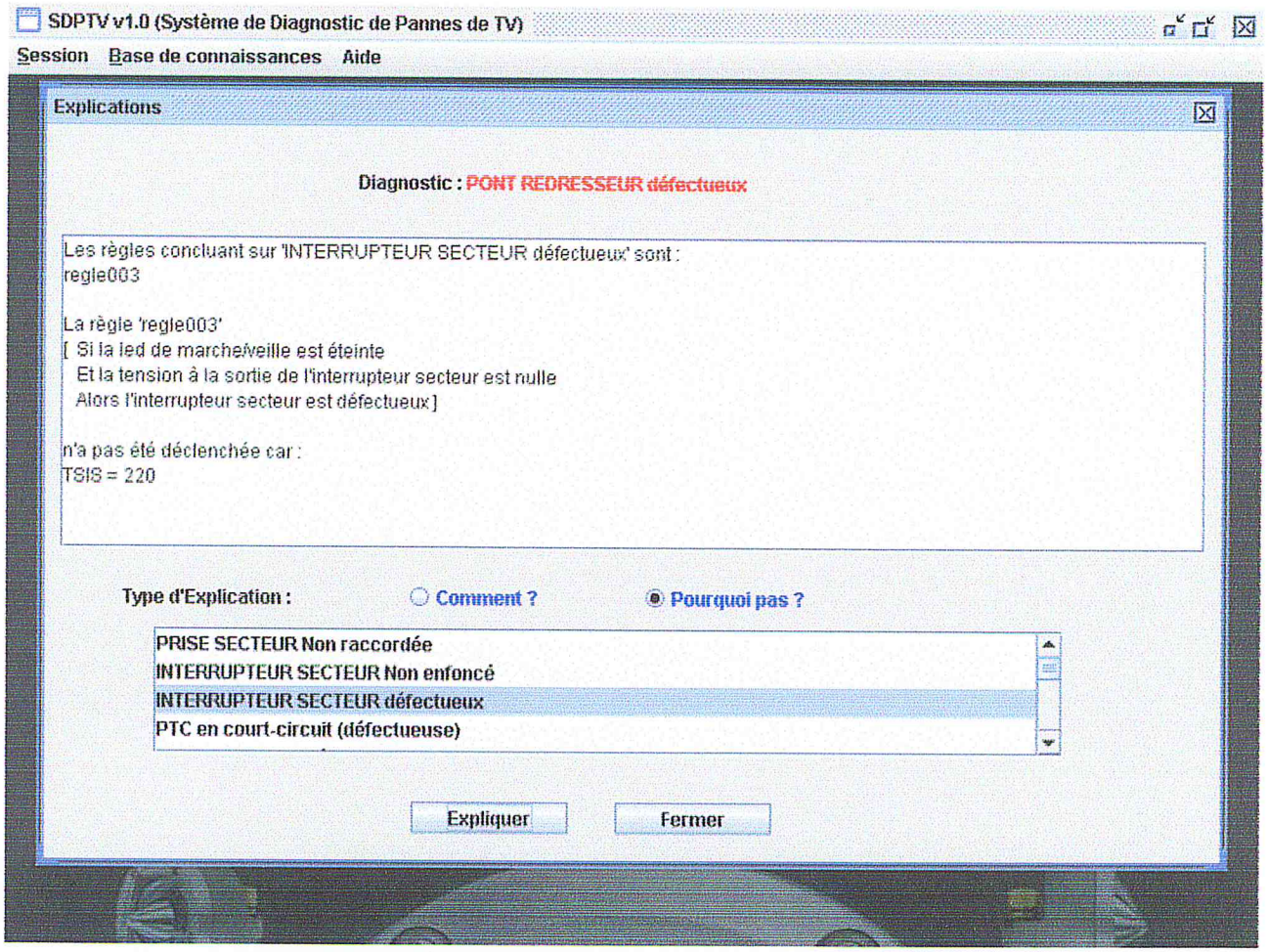


Figure III.42 – Exemple d'explication négative destinée à un utilisateur débutant

CONCLUSION GENERALE ET PERSPECTIVES

L'objet principal de ce travail a été de développer les capacités d'explication des systèmes experts (SE), en prenant comme exemple des systèmes experts construit autour d'une grammaire d'ordre 0+.

Le système SDPTV a été réalisé en JAVA et permet de trouver des solutions à différents problèmes de diagnostics et de pannes et bien sûr d'en expliquer le raisonnement. L'implantation de la connaissance sous forme de règles déclaratives permet, de par son caractère simple et uniforme, de rendre le système transparent à l'utilisateur. L'explication du raisonnement de SDPTV, constituée de la trace des règles déclenchées, offre à l'utilisateur une vision porteuse de sens du comportement du système et de ses connaissances, l'utilisateur accepte plus volontiers les conclusions qui lui sont proposées. Cependant, nous avons très vite senti les limites de l'explication dans ce type de système.

En effet, une première limitation est due en partie au raisonnement du SE qui est fondé sur un modèle simple : le moteur d'inférences exploite la base de règles, en utilisant le plus souvent un stratégie figée. La trace du raisonnement se présente alors sous la forme d'une arborescence, une forme qui reste très éloignée de toute structure d'explication humaine. Une deuxième limitation provient du fait que toutes les inférences ayant permis au SE de résoudre un problème ne sont pas forcément pertinentes, lors de l'explication. Par exemple, certaines règles de MYCIN font appel à des connaissances de simple bon sens ou correspondent à des définitions du domaine. Dans ce cas il ne sert à rien de réexpliquer à l'utilisateur quelque chose qu'il connaît déjà. D'autant que d'un raisonnement à l'autre, d'une session à l'autre, ces inférences vont se répéter de façon identique. De plus masquer ces inférences, lors de l'explication, permet de mettre en évidence les étapes importantes. Il est donc nécessaire que le SE sache faire preuve d'esprit de synthèse en présentant son raisonnement. Enfin, le raisonnement du SE correspond à celui de l'expert qui a enrichi la base de connaissances. Or l'expert, habitué à résoudre des problèmes, a tendance à aller vite sur certains cas, à utiliser des raccourcis de raisonnement, surtout si l'explication s'adresse à un utilisateur qui ne possède pas toute l'expérience requise.

Ces explications montrent bien qu'une simple présentation du raisonnement, une « explication trace », ne suffit pas (et ce, qu'elle que soit la qualité de la trace !).

Le système SDPTV n'est qu'à sa première version qui, bien entendu, doit évoluer. Parmi ces extensions que l'on se propose, certaines sont facilement réalisables avec le système sans modifications du moteur d'inférences ou de l'architecture interne du système, par contre d'autres extensions imposent la modification de ces dernières.

Les travaux réalisés nous conduisent à énoncer un certain nombre de perspectives :

- Améliorer l'ordre du système pour pouvoir utiliser des variables (ordre 1).
- Etendre la puissance d'expression des connaissances en utilisant des méta connaissances et des métarègles.
- Optimiser le cycle de base du moteur d'inférences, en offrant la possibilité d'emploi de métarègles pour la phase de résolution de conflits.

Annexe

Base de règles du système

Regle *Nom* = regle001 , *Priorite* = 5.
Si Prise_Secteur_Raccordée = Non
Alors Panne = PRISE SECTEUR Non raccordée
Explication : Si la prise secteur n'est pas raccordée Alors le TéléViseur ne peut démarrer

Regle *Nom* = regle002 , *Priorite* = 5.
Si Interrupteur_Secteur_Enfoncé = Non
Alors Panne = INTERRUPTEUR SECTEUR Non enfoncé
Explication : Si l'interrupteur secteur n'est pas enfoncé Alors le TéléViseur ne peut démarrer

Regle *Nom* = regle003 , *Priorite* = 5.
Si Led_M/V = Eteinte Et TSIS = 0
Alors Panne = INTERRUPTEUR SECTEUR défectueux
Explication : Si la led de marche/veille est éteinte Et la tension à la sortie de l'interrupteur secteur est nulle Alors l'interrupteur secteur est défectueux

Regle *Nom* = regle004 , *Priorite* = 5.
Si Led_M/V = Eteinte *Et* TCFCPA = 0 Et Fusible = Coupé
Alors PTC = suspect *Et* Alimentation = suspect
Explication : Si la led de marche/veille est éteinte Et la tension du condensateur de filtrage du circuit de l'alimentation primaire est nulle Et le fusible est coupé Alors la panne provient soit de la PTC soit du circuit de l'alimentation primaire

Regle *Nom* = regle005 , *Priorite* = 5.
Si PTC = suspect *Et* FC_après_remp_PTC_et_Fus = Non
Alors Panne = PTC en court-circuit (défectueuse)
Explication : Si la PTC est suspecte Et le fusible ne claque pas après dessoudage de la PTC, remplacement du fusible et mise en marche du TéléViseur Alors la PTC est en court-circuit (défectueuse)

Regle *Nom* = regle006 , *Priorite* = 5.
Si Alimentation = suspect *Et* FC_après_remp_Bu_et_Fus = Non
Alors Panne = TRANSISTOR BU défectueux
Explication : Si le circuit de l'alimentation primaire est suspect Et le fusible ne claque pas après dessoudage du transistor BU, remplacement du fusible et mise en marche du TéléViseur Alors transistor BU est défectueux

Regle *Nom* = regle007 , *Priorite* = 5.
Si Alimentation = suspect *Et* FC_après_remp_Bu_et_Fus = Oui
Alors Pont_redresseur = suspect
Explication : Si le circuit de l'alimentation primaire est suspect Et le fusible claque après dessoudage du transistor BU, remplacement du fusible et mise en marche du TéléViseur Alors le pont redresseur est suspect

Regle *Nom* = regle008 , *Priorite* = 5.
Si Pont_redresseur = suspect *Et* FC_après_remp_PR_et_Fus = Non
Alors Panne = PONT REDRESSEUR défectueux
Explication : Si le pont redresseur est suspect Et le fusible ne claque pas après dessoudage du pont redresseur, remplacement du fusible et mise en marche du TéléViseur Alors le pont redresseur est défectueux

Regle *Nom* = regle009 , *Priorite* = 5.
Si Pont_redresseur = suspect *Et* FC_après_remp_PR_et_Fus = Oui
Alors Panne = CONDENSATEUR de FILTRAGE défectueux
Explication : Si le pont redresseur est suspect Et le fusible claque après dessoudage du pont redresseur, remplacement du fusible et mise en marche du TéléViseur Alors le condensateur de filtrage du circuit de l'alimentation primaire est défectueux

Regle *Nom* = regle010 , *Priorite* = 5.
Si Led_M/V = Eteinte *Et* TCFCPA = 0 *Et* Fusible = Non coupé
Alors Filtre_secteur = suspect
Explication : Si la led de marche/veille est éteinte Et la tension du condensateur de filtrage du circuit de l'alimentation primaire est nulle Et le fusible n'est pas coupé Alors le filtre secteur est suspect

Regle *Nom* = regle011 , *Priorite* = 5.
Si Filtre_secteur = suspect *Et* Tension_après_filtre_sect = 220
Alors Panne = PONT REDRESSEUR défectueux
Explication : Si le filtre secteur est suspect Et la tension à la sortie du filtres secteur est égale à 220 Volts Alors le pont redresseur est défectueux

Regle *Nom* = regle012 , *Priorite* = 5.
Si Filtre_secteur = suspect Et Tension_après_filtre_sect = 0
Alors Panne = FILTRE SECTEUR défectueux
Explication : Si le filtre secteur est suspect Et la tension à la sortie du filtres secteur est nulle Alors le filtre secteur est défectueux

Regle *Nom* = regle013 , *Priorite* = 5.
Si Led_M/V = Eteinte *Et* TCFCPA >= 220 *Et* Tension_alim_sec = 0
Alors Circuit_intégré_com_déc = suspect
Explication : Si la Led de marche/veille est éteinte Et la tension du condensateur de filtrage du circuit de l'alimentation primaire est supérieure ou égale à 220 Volts Et la tension de l'alimentation secondaire est nulle Alors le circuit de commande de découpage est suspect

Regle *Nom* = regle014 , *Priorite* = 5.
Si Circuit_intégré_com_déc = suspect *Et* Tension_alim_circuit_intégré_com_déc >= 9
Alors Panne = CIRCUIT INTEGRE de COMMANDE de découpage défectueux
Explication : Si le circuit intégré de commande de découpage est suspect Et la tension d'alimentation du circuit intégré de commande de découpage est supérieure à 9 Volts Alors le circuit intégré de commande de découpage est défectueux

Regle *Nom* = regle015 , *Priorite* = 5.
Si Circuit_intégré_com_déc = suspect *Et* Tension_alim_circuit_intégré_com_déc < 9
Alors Panne = RESISTANCE de POLARISATION du circuit intégré de commande de découpage défectueuse
Explication : Si le circuit intégré de commande de découpage est suspect Et la tension d'alimentation du circuit intégré de commande de découpage est inférieure à 9 Volts Alors la résistance de polarisation du circuit intégré de commande de découpage est défectueuse

Regle *Nom* = regle016 , *Priorite* = 5.
Si Led_M/V = Clignote *Et* Tension_alim_aux_circuit_intégré_com_déc < 12
Alors Panne = CIRCUIT D'ALIMENTATION AUXILIAIRE du circuit intégré de commande de découpage défectueux
Explication : Si la led de marche/veille clignote Et la tension de l'alimentation auxiliaire du circuit intégré de commande de découpage est inférieure à 12 Volts Alors le circuit d'alimentation auxiliare du circuit intégré de commande de découpage est défectueux

Regle *Nom* = regle017 , *Priorite* = 5.
Si Led_M/V = Allumée_en_mode_veille *Et* TV_répond_à_clavier_télécommande = Non
Alors Panne = CIRCUIT de MISE en VEILLE défectueux
Explication : Si la led de marche/veille est allumée en mode veille Et la télévision ne répond ni au clavier ni à la télécommande Alors le circuit de mise en veille est défectueux

Regle *Nom* = regle018 , *Priorite* = 5.
Si Led_M/V = Eteinte *Et* écran_noir = Non *Et* Tension_alim_sec > 0 *Et* TV_répond_à_clavier_télécommande = Non
Alors Micro_contrôleur_gestion = bloqué
Explication : Si la led de marche/veille est éteinte Et l'écran n'est pas noir Et la tension de l'alimentation secondaire n'est pas nulle Et la télévision ne répond ni au clavier ni à la télécommande Alors le microcontrôleur de gestion est bloqué

Regle *Nom* = regle019 , *Priorite* = 5.
Si Micro_contrôleur_gestion = bloqué *Et* Tension_quartz = Continue
Alors Panne = QUARTZ du microcontrôleur de gestion défectueux
Explication : Si le microcontrôleur de gestion est bloqué Et la tension aux bornes du quartz est continue Alors le quartz du microcontrôleur de gestion est défectueux

Regle *Nom* = regle020 , *Priorite* = 5.
Si Micro_contrôleur_gestion = bloqué *Et* Tension_quartz = Alternative
Alors Panne = MICROCONTROLEUR de GESTION défectueux
Explication : Si le microcontrôleur de gestion est bloqué Et la tension aux bornes du quartz est alternative Alors le microcontrôleur de gestion est défectueux

Regle *Nom* = regle021 , *Priorite* = 5.
Si Led_M/V = S'allume_puis_s'éteint *Et* Tension_alim_sec = 0
Alors Une_des_tensions_au_sec_(U1,U2,..Un) = court_circuit
Explication : Si la led de marche/veille s'allume puis s'éteint Et la tension de l'alimentation secondaire est nulle Alors une des tensions au secondaire (U1,U2,..Un) est en court-circuit

Regle *Nom* = regle022 , *Priorite* = 5.
Si Une_des_tensions_au_sec_(U1,U2,..Un) = court_circuit *Et* Présence_Uk_k#i_après_dessoudage_Ui = Oui
Alors Panne = UN des ETAGES ALIMENTES PAR Ui est en court-circuit (défectueux)
Explication : Si une des tensions au secondaire (U1,U2,..Un) est en court-circuit Et présence de la tension Uk avec k#i après dessoudage de Ui Alors un des étages alimentés par Ui est en court-circuit

Regle *Nom* = regle023 , *Priorite* = 5.
Si Une_des_tensions_au_sec_(U1,U2,..Un) = court_circuit *Et* Présence_Uk_k#i_après_dessoudage_Ui = Non
Alors Panne = TRANSFORMATEUR de L'ALIMENTATION A DECOUPAGE défectueux
Explication : Si une des tensions au secondaire (U1,U2,..Un) est en court-circuit Et non présence de la tension Uk avec k#i après dessoudage de Ui Alors le transformateur de l'alimentation à découpage et défectueux

Regle *Nom* = regle024 , *Priorite* = 5.
Si Led_M/V = Allumée *Et* écran_noir = Oui *Et* Son = Présent *Et* Tension_Uligne_alim_THT < 115
Alors Panne = TRANSFORMATEUR de la THT défectueux
Explication : Si la led de marche/veille est allumée Et l'écran est noir Et le son fonctionne normalement Et la tension Uligne de l'alimentation de la THT est inférieure à 115 Volts Alors le transformateur de la THT est défectueux

Regle *Nom* = regle025 , *Priorite* = 5.
Si Led_M/V = Allumée *Et* Ligne_blanche_horiz_milieu_écran = Oui
Alors étage_vertical = suspect
Explication : Si la led de marche/veille est allumée Et présence d'une ligne blanche horizontale au milieu de l'écran Alors l'étage vertical est suspect

Regle *Nom* = regle026 , *Priorite* = 5.
Si étage_vertical = suspect *Et* Tension_alim_étage_vert = 0
Alors Panne = ALIMENTATION de L'ETAGE VERTICAL défectueuse
Explication : Si l'étage vertical est suspect Et la tension d'alimentation de l'étage vertical est nulle Alors l'alimentation de l'étage vertical est défectueuse

Regle *Nom* = regle027 , *Priorite* = 5.
Si étage_vertical = suspect *Et* Tension_alim_étage_vert = 25
Alors Panne = CIRCUIT INTEGRE de l'étage vertical défectueux
Explication : Si l'étage vertical est suspect Et la tension d'alimentation de l'étage vertical est égale à 25 Volts Alors le circuit intégré de l'étage vertical est défectueux

Regle *Nom* = regle028 , *Priorite* = 5.
Si Led_M/V = Allumée *Et* Défaut_linéarité_vert = Oui
Alors Panne = CIRCUIT de LINEARITE défectueux
Explication : Si la led de marche/veille est allumée Et un défaut de linéarité verticale est présent Alors le circuit de linéarité est défectueux

Regle *Nom* = regle029 , *Priorite* = 5.
Si Led_M/V = Allumée *Et* Instabilité_horiz_image = Oui
Alors Oscilateur_ligne = suspect
Explication : Si la led de marche/veille est allumée Et une instabilité horizontale de l'image est perçue Alors l'oscilateur ligne est suspect

Regle *Nom* = regle030 , *Priorite* = 5.
Si Oscilateur_ligne = suspect *Et* Réglage_fréq_ligne_stabilise_image = Non
Alors Panne = OSCILATEUR LIGNE (circuit intégré) défectueux
Explication : Si l'oscilateur ligne est suspect Et le réglage de la fréquence ligne ne stabilise pas l'image Alors l'oscilateur ligne (circuit intégré) est défectueux

Regle *Nom* = regle031 , *Priorite* = 5.
Si Oscilateur_ligne = suspect Et Réglage_fréq_ligne_stabilise_image = Oui
Alors Panne = mauvais ajustement de la FREQUENCE LIGNE
Explication : Si l'oscilateur ligne est suspect Et le réglage de la fréquence ligne stabilise l'image Alors c'était un problème d'ajustement de la fréquence ligne

Regle *Nom* = regle032 , *Priorite* = 5.
Si Led_M/V = Allumée *Et* Image = Noir_et_blanc
Alors Décodeur_PAL_SECAM = suspect
Explication : Si la led de marche/veille est allumée Et l'image est en noir et blanc Alors le décodeur PAL/SECAM est suspect

Regle *Nom* = regle033 , *Priorite* = 5.
Si Décodeur_PAL_SECAM = suspect *Et* Signal_SandCastle_correct = Oui
Alors Panne = Décodeur PAL/SECAM défectueux
Explication : Si le décodeur PAL/SECAM est suspect et le signal sandcastle est correct Alors le décodeur PAL/SECAM est défectueux

Regle *Nom* = regle034 , *Priorite* = 5.
Si Oscilateur_ligne = suspect *Et* Signal_SandCastle_correct = Non
Alors Panne = GENERATEUR du SIGNAL SANDCASTLE défectueux
Explication : Si le décodeur PAL/SECAM est suspect et le signal sandcastle n'est pas correct Alors le générateur du signal sandcastle est défectueux

Regle *Nom* = regle035 , *Priorite* = 5.
Si Led_M/V = Allumée *Et* Image = Dominante_magenta_pour_une_image_noir_et_blanc
Alors Panne = AMPLIFICATEUR VIDEO de la COULEUR VERTE défectueux
Explication : Si la led de marche/veille est allumée Et l'image est à dominante magenta pour une image en noir et blanc Alors l'amplificateur vidéo de la couleur verte est défectueux

Regle *Nom* = regle036 , *Priorite* = 5.
Si Led_M/V = Allumée *Et* Image = Dominante_jaune_pour_une_image_noir_et_blanc
Alors Panne = AMPLIFICATEUR VIDEO de la COULEUR BLEUE défectueux
Explication : Si la led de marche/veille est allumée Et l'image est à dominante jaune pour une image en noir et blanc Alors l'amplificateur vidéo de la couleur bleue est défectueux

Regle *Nom* = regle037 , *Priorite* = 5.
Si Led_M/V = Allumée *Et* Image = Dominante_cyon_pour_une_image_noir_et_blanc
Alors Panne = AMPLIFICATEUR VIDEO de la COULEUR ROUGE défectueux
Explication : Si la led de marche/veille est allumée Et l'image est à dominante cyon pour une image en noir et blanc Alors l'amplificateur vidéo de la couleur rouge est défectueux

Regle *Nom* = regle038 , *Priorite* = 5.
Si Led_M/V = Allumée *Et* Son = Absent *Et* Réglage_volume_fonctionnel = Oui
Alors Amplificateur_audio = suspect
Explication : Si la led de marche/veille est allumée Et le son ne fonctionne pas Et le réglage du volume est fonctionnel Alors l'amplificateur audion est suspect

Regle *Nom* = regle039 , *Priorite* = 5.
Si Amplificateur_audio = suspect *Et* Haut_parleur_opérationnel = Oui
Alors Panne = CIRCUIT INTEGRE de L'AUDIO défectueux
Explication : Si l'amplificateur audio est suspect Et le haut parleur est opérationnel Alors le circuit intégré du son est défectueux

Regle *Nom* = regle040 , *Priorite* = 5.
Si Amplificateur_audio = suspect *Et* Haut_parleur_opérationnel = Non
Alors Panne = HAUT PARLEUR défectueux
Explication : Si l'amplificateur audio est suspect Et le haut parleur n'est pas opérationnel Alors le haut parleur est défectueux

Regle *Nom* = regle041 , *Priorite* = 5.
Si Led_M/V = Allumée *Et* Son = Absent *Et* Réglage_volume_fonctionnel = Non
Alors Panne = MICROCONTROLEUR de GESTION défectueux
Explication : Si la led de marche/veille est allumée Et le son ne fonctionne pas Et le réglage du volume n'est pas fonctionnel Alors le microcontrôleur de gestion est défectueux

Regle *Nom* = regle042 , *Priorite* = 5.
Si Led_M/V = Allumée *Et* Son = Absent_mode_TV_et_présent_mode_peritel
Alors Panne = ETAGE FI SON défectueux
Explication : Si la led de marche/veille est allumée Et le son est absent en mode TV et présent en mode péritel Alors l'étage FI son est défectueux

Regle *Nom* = regle043 , *Priorite* = 5.
Si Led_M/V = Allumée *Et* TV_fonctionne_avec_télécommande = Non *Et* Télécommande_fonctionne_avec_autre_TV_ou_testeur_télécommande = Oui
Alors Panne = RECEPTEUR INFRAROUGE défectueux
Explication : Si la led de marche/veille est allumée Et la télévision ne fonctionne pas avec la télécommande Et la télécommande fonctionne avec une autre télévision du même type ou avec un testeur de télécommandes Alors le récepteur infrarouge est défecteux

Regle *Nom* = regle044 , *Priorite* = 5.
Si Led_M/V = Allumée *Et* TV_fonctionne_avec_télécommande = Non *Et* Télécommande_fonctionne_avec_autre_TV_ou_testeur_télécommande = Non
Alors Panne = TELECOMMANDE défectueuse
Explication : Si la led de marche/veille est allumée Et la télévision ne fonctionne pas avec la télécommande Et la télécommande ne fonctionne pas avec une autre télévision du même type ou avec un testeur de télécommandes Alors la télécommande est défectueuse

Regle *Nom* = regle045 , *Priorite* = 5.
Si Led_M/V = Allumée *Et* Mémorisation_chaînes = Non
Alors Panne = MEMOIRE EEPROM défectueuse
Explication : Si la led de marche/veille est allumée Et la mémorisation des chaînes ne fonctionne pas Alors la mémoir EEPROM est défectueuse

Regle *Nom* = regle046 , *Priorite* = 5.
Si Led_M/V = Allumée *Et* Une_bande_grise_défile_verticalement = Oui
Alors Panne = CONDENSATEUR de FILTRAGE de l'alimentation primaire défectueux
Explication : Si la led de marche/veille est allumée Et une bande grise défile verticalement Alors le condensateur de filtrage de l'alimentation primaire est défectueux

Regle *Nom* = regle047 , *Priorite* = 5.
Si Led_M/V = Allumée *Et* Changement_incontrôlé_chaînes = Oui
Alors Panne = BUS I2C défectueux
Explication : Si la led de marche/veille est allumée Et un changement incontrôlé des chaînes est perçu Alors le bus I2C est défectueux

Regle *Nom* = regle048 , *Priorite* = 5.
Si Led_M/V = Allumée *Et* Réception_se_bloque_sur_une_chaîne = Oui
Alors Panne = BUS I2C défectueux
Explication : Si la led de marche/veille est allumée Et la réception se bloque sur une chaîne Alors le bus I2C est défectueux

Bibliographie

[Alliot et al 02] J.-M. Alliot, T. Schiex, P. Brisset, F. Garcia,
Intelligence artificielle et informatique théorique 2ème édition,
Ed. Cepadues, 2002.

[Alty et al 86] J.L. Alty, M.J. Coombs.
Systèmes experts : concepts et exemples.
Ed. Masson, 1986.

[Baker et al 00a] M. Baker, M. Joab, B. Safar, D. Schlienger,
Etudes d'explications dans un corpus de dialogues finalisés,
Psychologie de l'interaction, numéro spécial, n°9-10, p. 179-210, 2000.

[Benchimol et al 90] G. Benchimol, P. Lévine, J.-C. Pomerol,
Systèmes experts dans l'entreprise,
Ed. Hermès, 1990.

[Bonnet 84] A. Bonnet,
L'intelligence artificielle : Promesses et Réalités,
Ed. InterEditions, 1984.

[Buchanan et al 84] B.G. Buchanan, E.H. Shortilffe,
Rule-based expert systems : the MYCIN experiments of the Stanford heuristic programming project,
Ed. Addison-Wesley, 1984.

[Cawsey 89] A. Cawsey,
Generating Explanatory Discourse : a plan-based, interactive approach,
PhD Thesis, University of Edinburgh, 1989.

[Chandrasekaran et al 88] B. Chandrasekaran, M.C. Tanner, J.R. Josephson,
Explanation: the role of strategies and deep models,
in Handler JA (ed). Expert systems: the user interface.
Ablex, Norwood, NJ, 1988.

[Clancey et al 81] W.J. Clancey, R. Letsinger,
NEOMYCIN : Reconfiguring a rule-based expert system for application to teaching,
IJCAI7, p 828-836, 1981.

[Clancey 82] W. Clancey,
GUIDON,
in the handbook of artificial intelligence, p. 267-278, William Kauffman Inc., Los altos, 1982.

[Collins 92] H.M. Collins,
Experts artificiels, Seuil, Paris, 1992.

[Console et al 88] L. Console, P. Torasso,
Heuristic and causal reasoning in medical diagnosis,
in proceding of the AAAI symposium on artificial intellegence in medecine, Stanford, 1988.

[Ermine 89] J.-L. Ermine.
Systèmes experts.
Ed. Techniques et Documentation Lavoisier, 1989.

[Ermine 93] J.-L. Ermine,
Génie Logiciel et Génie Cognitif pour les systèmes à base de connaissances,
Ed. Techniques et Documentation Lavoisier, 1993.

[Farreny 85] H. Farreny,
Les systèmes experts : principes et exemples,
Ed. Cepadues, 1985.

[Farreny 89] H. Farreny.
Les systèmes experts : principes et exemples,
Ed. Berti, 1989.

[Ganet 03] L. Ganet,
Un modèle pour la génération d'explication basé sur la catégorisation contextuelle,
Symposium TAC (Tâche, Activité et Contexte), 2003.

[Harmon et al 88] P. Harmon, D. King,
Systèmes experts professionnels : conception et implémentation.
Ed. Masson, 1988.

[Hasling et al 84] D.W. Hasling, W.J. Clancey, G. Rennels,
Explanations for a diagnostic consultation system,
International journal of Man-Machine studies 20, pp.3-19, 1984.

[Jiménez 90] C. Jiménez,
Sur l'explication dans les systèmes à base de règles : le système PROSE,
Thèse de doctorat de l'université de Paris VI, 1990.

[Karkan et al 93] J.-M. Karkan, G.Tjoen,
Systèmes experts : un nouvel outil pour l'aide à la décision,
Ed. Masson, 1993.

[Kassel 86] G. Kassel,
Le système d'explication CQFE, une forme de métaraisonnement intégrant règles et objets,
Thèse de doctorat de l'université d'Orsay, 1986.

[Lauriere 82] J.L. Lauriere,
Représentation et utilisation des connaissances.2ème partie : Représentation des connaissances,
Techniques et science informatique (TSI) : vol 1 n°2, 1982.

[Lemaire 92] B. Lemaire,
Construction et transmission d'explication dans les systèmes à base de connaissances,
Thèse de doctorat de l'université de Paris-Sud, 1992.

[Lund 03] K.S. Lund ,
Analyse de l'activité explicative en interaction : Etudes de dialogues d'enseignants de physique en formation interprétant les interactions entre élèves,
Thèse de doctorat de l'université Joseph Fourier, 2003.

[Mckeown 85] K.R. Mckeown,
Discourse strategies for generating natural language text,
in artificial intelligence 27, p. 1-42, 1985.

[Moore et al 89] J.D. Moore, W.R. Swartout,
A reactive approach to explanation,
in proceedings of the 11th IJCAI, Detroit, p. 1504-1510, 1989.

[Neches et al 84] R. Neches, W.R. Swartout, J.D. Moore,
Enhanced maintenance and explanation of expert systems through explicit models of their development,
in IEEE Workshop on principles of knowledge-based systems, Denver, 1984.

[Newell 81] A. Newell,
The knowledge level,
In AI Magazine 2 (2), p. 1-20, 1981.

[Paris et al 88] C.L. Paris, M.R. Wick, W.B. Thompson,
The line of reasoning versus the line of explanation,
in Proceedings of the AAAI workshop on explanation, st-Paul, 1988.

[Rich 87] E. Rich,
Intelligence Artificielle.
Ed. Masson, 1987.

[Richards 98] D.C. Richards,
The reuse of knowledge in ripple down rule knowledge based systems,
PhD, University of New South Waltes, Sydney, Australia, 1998.

[Richards 03] D.C. Richards,
Knowledge-based system explanation: the ripple-down rules alternative,
Knowledge and Informations Systems 5, p. 2-25, 2003.

[Rousset 90] M.C. Rousset,
Diagnosis systems and explanations,
in rapport interne LRI, n° 588, 1990.

[Safar 87] B. Safar,
Le problème des explication négatives dans les systèmes experts : le système POURQUOI-PAS ?
Thèse de doctorat de l'université de Paris XI, 1987.

[Saïdi 91] M. Saïdi,
Aplusix : Evaluation progressive de la connaissance et planification dans un environnement d'apprentissage de l'algèbre,
in Actes des deuxièmes journées EIAO de Cachan , 1991.

[Saïdi 92] M. Saïdi,
Planification et explication du raisonnement d'un résolveur complexe en algèbre : application aux factorisations de polynômes et aux résolveurs d'équations,
Thèse de doctorat de l'université de Paris XI, 1992.

[Sawyer et al 88] B. Sawyer, D. Foster,
Programmation des systèmes experts en PASCAL,
Ed. Masson, 1988.

[Shortliffe 74] E.H. Shortliffe,
MYCIN : a rule-based computer program for advising physicians regarding antimicrobial therapy selection,
PhD disertation, Stanford University, 1974.

[Soury 98] S. Soury-Lavergne,
Etayage et explication dans le préceptorat distant, le cas de TéléCabri,
Thèse de doctorat de l'université Joseph Fourier, 1998.

[Swartout 83] W.R. Swartout,
XPLAIN : a system for creating and explaining expert consulting programs,
Artificial Intelligence 21 (3), pp. 285-325, 1983.

[Swartout 85] W.R. Swartout,
Explanation and the role of the user model: how much will it help?,
in User Modelling Panel, 9th IJCAI, pp.1298-1302, 1985.

[Voyer 87] R. Voyer,
Moteurs des systèmes experts,
Ed. Eyrolles, 1987.

[Weiner 80] J.L. Weiner,
BLAH, a system which explains its resoning,
Artificial intelligence 15 (1-2), pp.19-48, 1980.