Intellectica, 1991/2, 12, pp. 101-137

Jean-Paul KRIVINE et Jean-Marc DAVID

L'ACQUISITION DES CONNAISSANCES VUE COMME UN PROCESSUS DE MODÉLISATION : MÉTHODES ET OUTILS

- *Generic Tasks*
- RIME

Conclusions

Introduction

"L'acquisition des connaissances est une des difficultés majeures de la conception des systèmes experts". Combien de centaines d'articles commencent par une telle introduction. Pourtant cette affirmation, bien qu'exacte, est trop générale. Il nous semble important de mieux préciser où est la difficulté et quelle est la nature du "goulet d'étranglement", pour reprendre l'expression consacrée.

C'est ce que nous allons chercher à faire dans la première partie de cet article en montrant comment l'acquisition des connaissances, perçue initialement comme un problème de *transcription,* est maintenant considérée comme un problème de *modélisation*. Nous décrirons ensuite les travaux visant à donner une base à cette modélisation. Nous montrerons enfin comment il est alors possible de mettre au point des *méthodes de conception* et de construire des *outils automatiques* d'aide à l'acquisition des connaissances. Les affirmations suivantes, que nous chercherons à argumenter, vont servir de structure à cet article :

1- L'acquisition des connaissances est principalement un processus de *construction de modèles.*

2- La mauvaise adéquation du niveau d'abstraction des formalismes classiquement utilisés dans la première génération de systèmes experts a induit un processus d'acquisition des connaissances *dirigé par l'implémentation*.

3- Les méthodes de conception dites "orientées objets" permettent de bien modéliser une partie du problème (modélisation du domaine) en proposant

un niveau d'abstraction adapté. Mais elles sont incomplètes pour modéliser la résolution du problème dans son ensemble.

4- Construire un modèle de la résolution de problèmes nécessite des primitives permettant de représenter le raisonnement et les méthodes de résolution.

5- Des outils d'aide et des méthodes de conception assez puissants vont pouvoir s'appuyer sur ce paradigme. Ces outils seront exposés dans le dernier paragraphe de cet article.

I. Transcription ou modélisation ?

L'acquisition des connaissances constitue l'une des tâches les plus délicates lors de la conception d'un système à base de connaissances. La littérature en ce domaine est abondante. Mais avant de poursuivre, il importe de préciser un point de vocabulaire. Les travaux dans ce domaine se sont clairement divisés en deux groupes ayant des objectifs relativement différents (Aussenac 1989) : le *transfert d'expertise* qui vise à mettre en place des moyens pour donner à un système à base de connaissances les connaissances issues d'un expert, de documents ou de manuels, et l'*apprentissage automatique*, qui vise à doter les systèmes de capacités d'acquisition de nouvelles connaissances par eux-mêmes. Dans cet article, nous nous intéresserons essentiellement au transfert d'expertise dans le sens donné ci-dessus.

Mais en quoi consiste donc l'acquisition des connaissances lors de la conception d'un système expert[1]? Il est important de souligner que la définition de cette activité a largement évolué dans la dernière décennie. L'acquisition des connaissances a d'abord été perçue comme une activité d'accès à quelque-chose de préexistant puis de *transcription* dans un formalisme donné. Ainsi, l'expert du domaine était sensé posséder une connaissance plus ou moins explicite qu'il s'agissait "*d'extraire*" (d'où

[1] Les termes *système expert* et *système à base de connaissances* seront employés sans distinction dans la suite de l'article

l'expression “extraction des connaissances”) pour ensuite la transcrire dans un programme, selon un formalisme plus ou moins universel : règles de productions, objets, logique... (Hayes-Roth & al. 1983). Très tôt, des difficultés sont apparues.

D'une part, le décalage trop important entre le langage utilisé par l'expert et le niveau d'abstraction des formalismes de représentation des connaissances a constitué un obstacle que de nombreux travaux ont cherché à surmonter. Ceux-ci se sont principalement attachés à fournir des niveaux intermédiaires permettant une meilleure modélisation du problème. Nous reviendrons largement sur ces travaux qui ont conduit à redéfinir l'acquisition des connaissances en terme de construction de modèles.

D'autre part, le caractère “subconscient” et implicite des connaissances de l'expert a nécessité le développement de nombreuses techniques de verbalisation et d'aide aux interviews d'experts. A propos de ces techniques “d'élicitation”, on pourra se reporter à Hoffman (1989) ou Aussenac (1989). Bien que ceci ne soit pas totalement indépendant de notre propos, nous n'insisterons pas sur cet aspect du problème. Mais notons avec Breuker & Wielenga (1989) que le problème n'est pas tant qu'il serait impossible d'obtenir par de telles techniques une description très détaillée de toutes les connaissances utilisées, mais que nous serions incapables de les organiser. Le goulet d'étranglement (la phase d'acquisition des connaissances) apparaît ainsi plutôt comme un facteur limitatif, un filtre indispensable, qui permet *effectivement* d'aborder le problème. Ces méthodes de verbalisation, pour aussi utiles qu'elles soient, ne peuvent donc se suffire.

Mais revenons sur la première difficulté, le décalage entre le langage dans lequel l'expert exprime ses connaissances et le niveau d'abstraction des formalismes classiques de représentation des connaissances. Il est apparu progressivement que le problème n'était pas simplement un problème de transcription, mais fondamentalement un problème de

modélisation (Boose, Nagai 1987). L'acquisition des connaissances se fixe alors pour but la construction de modèles (Clancey 1985, Wielenga & Breuker 1986, Clancey 1989).

En effet, si les systèmes que l'on construit visent à simuler des performances d'experts dans un domaine particulier, pour une tâche précise, ils n'ont pas la prétention de refléter ou modéliser les processus réellement mis en oeuvre par le cerveau humain. A partir des données et connaissances issues des interviews et des verbalisations, le cogniticien doit *organiser* cette information et *abstraire* des concepts. En ce sens, il s'agit bien de *construire* un modèle.

Il importe néanmoins de préciser que cette modélisation comporte deux aspects imbriqués : une modélisation du domaine et une modélisation du processus de résolution, du raisonnement (Wielinga & Breuker 1986, Steels 1990). Si le premier a souvent été largement traité, en particulier au travers des “méthodes de conception orientées objets”, le second est souvent négligé ou ramené dans le formalisme du premier. Pourtant, et c'est un des propos essentiels de cet article, la modélisation explicite des processus de raisonnement est indispensable pour la mise en place de méthodes complètes de conception ou de création d'outils automatiques. Elle nécessite des primitives et un formalisme adapté de représentation. Au delà d'une grande diversité de vocabulaire, de nombreux travaux ont recherché une telle modélisation à ce que Newell a appelé le “knowledge level” (1981).

II. Un peu d'histoire

Comme nous l'avons indiqué, l'acquisition des connaissances, encore au début des années 80, était comprise comme un processus de transcription dans un formalisme réputé général (souvent basé sur un système de règles de productions) d'un savoir recueilli auprès d'un expert. Ainsi, dans “Building Expert Systems”, l'ouvrage de référence du début des années 80,

F. Hayes-Roth, D. Waterman et D. Lenat distinguaient-ils 5 phases dans la conception d'un système expert.

[1] Identification : Déterminer les caractéristiques du problème.
[2] Conceptualisation : Trouver les concepts représentant les connaissances.
[3] Formalisation : Concevoir les structures pour organiser les connaissances.
[4] Implémentation : Écrire les règles qui capturent les connaissances.
[5] Test : Valider les règles qui capturent les connaissances.

*figure 1 **:** Les étapes de la construction d'un système expert (Hayes-Roth & al. 1983)*

L'étape 3 de formalisation était ainsi décrite :

> *Formalization involves mapping the key concepts and relations into a formal representation suggested by some expert-system building tool or language. The knowledge engineer must select the language and, with the help of the expert, represent the basic concepts and relations within the language framework* (Hayes-Roth & al. 1983).

Pour caractériser cette époque, plusieurs auteurs parleront d'*acquisition des connaissances dirigée par l'implémentation* (Hickman & al. 1989). En effet, le niveau d'abstraction fourni par les structures classiques de représentation des connaissances, les “expert-system building tools” cités plus haut (frames, logique, règles, etc.) est trop bas pour permettre de représenter les connaissances et le contrôle de manière satisfaisante. Chandrasekaran ira même jusqu'à employer l'expression “*langage d'assemblage des connaissances*” pour comparer ce niveau à celui effectivement nécessaire :

> *These architectures (with relatively small additions, if needed) are computationaly universal. Thus the important point about building*

> *knowledge systems with them is not whether a task can be performed, but whether they offer knowledge and control constructs that are natural to the task. All of these (and other similar) languages fall short when one considers tasks of some complexity such as planning or diagnosis. (...) The level of abstraction of these languages obscurses the essential nature of the information processing that is needed for the performance of higher level tasks. They are like knowledge system assembly languages, rather than programming languages with constructs for capturing the essence of the information processing phenomena* (Chandrasekaran 1987).

Ainsi, dans la réalité, les étapes de formalisation et d'implémentation d'un système expert vont bien souvent faire appel à des “trucs” et astuces de programmation. La figure 2 tirée de *Programming Expert Systems in OPS5* (Bronstown & al 1985) récapitule les techniques pour “bien” programmer en OPS5. Un tel type de connaissance est en fait indispensable, et pas seulement dans le but de “programmer efficacement”. On est loin de “*l’essence des processus de traitement de l'information”* souhaitée par Chandrasekaran.

Comprendre le fonctionnement d'un système expert devient alors une tâche ardue. La figure 3 montre un exemple de règle de MYCIN et les connaissances nécessaires à son interprétation. Cet exemple est tiré de Farenny (1987).

[1] Avoid conditions that match many working memory elements.

a) Add attribute tests that change the condition so that fewer working memory elements match it. Tests may be based on restrictions to data or on state of processing.
b) Add new element classes or modify representation to change the data elements so that fewer of them match the condition.
c) Represent and enumerate sequences carefully.

[2] Avoid big cross-products between conditions.

a) Order the conditions so the more restrictive ones occur first. This limits the number of consistent matches that are passed on to the next condition.
b) Write rules with a few big conditions rather than many simple conditions by merging the attributes of related element classes into fewer element classes.
c) Use “build” to specialize rules.

[3] Avoid frequent changes to matched conditions.

a) Put conditions that match frequently changing elements as far toward the end of the rule as possible.
b) Avoid excessive changes in control elements.

[4] Make matching individual condition elements faster.

a) Put the most restrictive attribute tests first to speed the match of working memory elements against conditions.
b) Change the representation of data to speed up matching.

[5] Limit the size of the conflict set.

[6] Call user-defined functions.

figure 2 : Programmer efficacement en OPS5 (Brownston & al. 1985)

Cette difficulté a également été largement soulignée à propos de l'évolution du système R1 de Digital Equipement, système considéré comme l'un des touts premiers systèmes experts opérationnels (Bachant & McDermott 1984).

La règle suivante est issue du système MYCIN, pour le diagnostic des maladies du sang. MYCIN est l'un des premiers systèmes experts. Son architecture est représentative d'une grande partie des réalisations actuelles.

R1 SI la coloration de l'organisme est GRAM+
et si la morphologie de l'organisme est Cocci
et si le mode de développement de l'organisme est en colonies
ALORS
il existe une évidence (0.7) que l'identité de l'organisme soit Staphylococcus.

Pour écrire une telle règle, le concepteur devra savoir comment MYCIN fonctionne. Ici, il faut savoir que cette règle, parce qu'elle conclut sur l'identité de Staphylococcus et parce que le moteur fonctionne en chaînage arrière, ne sera envisagée que lorsque le système se préoccupera de déterminer l'identité d'un organisme. Il faut de plus savoir qu'à chaque instant, le moteur s'intéresse à un “contexte” particulier, c'est-à-dire un organisme à retrouver. Un ordre des prémisses dans une autre règle fera en sorte que MYCIN s'intéressera d'abord à l'organisme courant avant de rechercher des organismes présents antérieurement. A chaque type d'organisme est associé un ensemble de caractéristiques (par exemple, identité de l'organisme pour l'organisme courant). A chaque caractéristique est associé un attribut “mis-à-jour-par” qui contient la liste des règles qui permettent d'établir la caractéristique en question (l'identité de l'organisme, par exemple). La règle R1 ci-dessus fait partie de cette liste de règles...
Voilà pourquoi et comment cette règle va être examinée !

figure 3 : Un contrôle largement implicite

A la base de ceci était sans doute la conviction que les langages de l'IA sont suffisamment puissants et naturels pour encoder les connaissances utiles. N. Aussenac développe à ce propos le parallèle entre différents modèles cognitifs et différents formalismes de représentation des connaissances (Aussenac 1989). Porcheron, dans sa thèse, rappelle la

problématique des règles de production et "*l'opinion sur la qualité anthropomorphique des systèmes de production"* (Porcheron 1990). Il cite à ce propos Newell et Simon :

> *We confess to a strong premonition that the organization of human programs closely resembles the production system organization...* (Newell & Simon 1972).

Aujourd'hui, sans aller jusque là, on observe toujours une tendance à attribuer des qualités intrinsèques très fortes au différents langages de représentation des connaissances. Si la "mode" règles de production est maintenant largement atténuée, certaines des critiques faites peuvent s'appliquer aux langages objets et "méthodes orientées objets". Nous y reviendrons dans le paragraphe suivant.

Les limites des formalismes classiques et le cadre contraignant qu'ils imposent s'ils sont adoptés comme seul formalisme de conceptualisation ont été soulignés très tôt. Ainsi, il est intéressant de rapporter le commentaire fait par Davis lui-même sur Mycin à l'occasion de la conception de TEIRESIAS, système d'aide à l'acquisition de nouvelles règles de MYCIN :

> *We speculated about the possibility of a representation of control structures that emphasized intentional information, which, combined with a formalization of the concept of explanation, might make the system far more flexible and general. It might, ideally, be possible for the system to examine its own code, generating explanations of what it found there* (Davis & Lenat 1982).

Néanmoins, Clancey est probablement le premier à réaliser un système sur de tels principes. NEOMYCIN, qui traite le même problème que MYCIN, va manipuler des primitives telles que "grouper et différencier",

“identifier le problème”, “établir l'espace des hypothèses” (Clancey 1986) alors que MYCIN faisait du “chaînage arrière”, du “pattern matching”, de la “résolution de conflits”. Ce type d'architecture va être à la base d'une nouvelle génération de systèmes experts appelés “Systèmes Experts de Seconde Génération” (David & al. 1989), et va offrir un cadre pour des méthodes plus élaborées d'acquisition des connaissances.

III. Modéliser le domaine, modéliser le raisonnement

1. A propos des méthodes de conception orientées objets

Avec le large développement des langages orientés objets est apparue une multitude de *méthodes de développement orientées objets*.

Il nous semble important de les examiner avec un regard critique afin d'éviter de retomber dans les travers énoncés plus haut à propos des systèmes à base de règles. Nous allons préciser leurs forces et leurs limites.

Classiquement, la conception d'un système à base de connaissances, la construction d'un *modèle qualitatif* selon l'expression de Clancey, comportera deux aspects (d'ailleurs largement interdépendants) :

- Une modélisation du domaine : “domain models” dans Wielinga & Breuker (1986) et Steels (1990).

- Une modélisation du processus de résolution, du raisonnement : “task level” dans Wielinga & Breuker (1986), “Role-limiting-methods” dans McDermott (1988) ou “Problem Solving Methods” pour d'autres auteurs.

Le premier type de modélisation va tout naturellement faire appel à un certain nombre de primitives. On cherchera à décrire les *objets*, les entités du domaine, à identifier les *concepts*. Chacun d'entre eux sera qualifié en

termes de *caractères*, de *comportements*. On cherchera ensuite à décrire leurs *relations* mutuelles et les décompositions éventuelles. Ainsi, tout logiquement, les langages orientés objets (ou des langages analogues, langages centrés relations par exemple) vont offrir un cadre naturel pour ce type de modélisation. Ils offrent en effet les bonnes primitives de modélisation : classes, attributs, héritage, relations, méthodes, etc. Ils apparaissent comme des bons formalismes de représentation, au bon niveau d'abstraction, pour le premier type de modélisation évoqué ci-dessus.

Les méthodes orientées objets ne font que formaliser cette adéquation en proposant un protocole permettant l'émergence des bons concepts. Ainsi, G. Booch (1986) propose dans sa méthode la décomposition suivante :

- Identifier les objets du monde réel, les caractéristiques de ces objets. (Différentes techniques sont suggérées pour aider à les faire ressortir à partir de la verbalisation des experts).
- Identifier les opérations : ce sont les actions que peut subir ou provoquer chaque objet.
- Établir la visibilité : comment chaque objet est vu des autres, comment ils communiquent.
- Établir l'interface, c'est-à-dire la visibilité des objets par le monde extérieur.
- Implémenter les objets, ce qui peut nécessiter la création d'objets de plus bas niveau d'abstraction.

Cette décomposition assez caractéristique se retrouve dans beaucoup d'autres méthodes (voir par exemple Meyer 1988).

Mais soulignons ici le point essentiel : la force de ce type de méthodologie provient d'abord de la *bonne adéquation* entre le formalisme

de représentation des connaissances (le concept d'objet) et la réalité que l'on veut modéliser (le domaine).

Les limites sont précisement celles du type de modélisation que l'on aborde. Vouloir donner un caractère plus général à ces méthodes, et en particulier celui de permettre les modélisations du raisonnement, conduit aux mêmes biais que ceux énoncés plus haut à propos des “débuts” de l'acquisition des connaissances.

En effet, modéliser un raisonnement, un processus de résolution de problème va nécessiter d'*autres primitives*. De ce point de vue, les “méthodes” (au sens des langages objets) apparaissent aussi pauvres que les règles. L'adéquation soulignée ci-dessus disparait. Les méthodes dites “orientées objets” ne suffisent plus.

Enfin, il convient de répéter ici que “modélisation du domaine” et “modélisation du raisonnement”, sont totalement imbriqués. On ne modélise jamais un domaine dans l'absolu, sans la définition d'un traitement spécifique, d'une utilisation particulière. La “bonne adéquation” mentionnée plus haut concerne donc plutôt des problèmes où le traitement sur les données est relativement simple et homogène, ce qui est souvent le cas dans les systèmes d'information classiques.

2. Des primitives pour modéliser le raisonnement

(i) Generic Tasks de Chandrasekaran

De nombreux auteurs ont souligné la mauvaise adéquation des formalismes de description de la première génération de systèmes experts (règles, objets, frames, etc). Ainsi Chandrasekaran fera-t-il remarquer que MYCIN est soit décrit comme un système de diagnostic des maladies du sang, ce qui est exact mais trop général, soit comme un système expert faisant du chaînage arrière et utilisant des coefficients de vraisemblance, ce qui est également vrai, mais peu informatif. Pourtant, on conçoit qu'il existe des stratégies différentes entre, par exemple, la méthode de

conception d'une automobile et la manière dont on va diagnostiquer une panne de moteur. Or, ces deux tâches vont utiliser les mêmes primitives : déclencher des règles, résoudre un ensemble de conflits, vérifier des prémisses, etc.

A l'opposé, diagnostic médical et diagnostic de cartes électroniques par exemple, sont deux tâches de diagnostic. Mais chacune met en oeuvre des techniques et méthodes très différentes. Le terme “diagnostic” décrivant ces deux activités apparaît ainsi comme un peu trop général et ne précise pas la méthode mise en oeuvre.

D'un côté donc, des langages de trop bas niveau d'abstraction, d'un autre côté, une description trop générale Chandrasekaran (1983, 1987), Chandrasekaran & al. (1989).

L'objectif est alors double. D'une part, on va chercher à fournir un langage de description au bon niveau d'abstraction et d'autre part, un ensemble de modèles génériques pouvant servir de briques de base à cette modélisation vont être proposés.

Il s'agit en fait de trouver le pendant des primitives de modélisation du domaine, évoquées dans le paragraphe précédent (objets, relations, attributs, etc.), mais cette fois pour la modélisation du raisonnement. Le bien fondé de cette approche est conforté par l'identification, dans des systèmes différents, de tâches similaires, par exemple l'abstraction de données et la classification dans NEOMYCIN (Clancey 1986), MDX2 (Sticklen 1989) ou DIVA (David & Krivine 1989).

C'est dans ce cadre que Chandrasekaran a proposé son approche “Tâches Génériques” sur laquelle nous reviendrons.

(ii) Le modèle d'expertise de KADS

Le développement de KADS (Breuker & Wielenga 1985, 1989) s'appuie sur des motivations similaires mais avec de plus, la volonté de conserver ce qui était sain dans les méthodes classiques de développement de logiciels, tout en y intégrant la spécificité des systèmes à base de

connaissances (Hickman & al. 1989). Ces spécificités concernent essentiellement la *phase d'analyse* qui va déboucher sur des modèles plus complexes. Les auteurs ont ainsi proposé un *modèle d'expertise* permettant de décrire la résolution du problème par le système. C'est le modèle en 4 couches proposé par (Wielinga & Breuker 1986) où chaque couche va organiser et structurer la couche inférieure :

[1] *Domaine :* Connaissances statiques décrivant les concepts, relations et structures du domaine.
[2] *Inférence :* Les inférences qui peuvent être faites sur les entités du niveau précédent (description en termes de rôles et de sources de connaissances).
[3] *Tâches :* Comment effectuer et séquencer les inférences afin d'atteindre un but.
[4] *Stratégie :* comment enchaîner les différentes tâches, gérer les échecs d'une méthode de résolution, etc.

On retrouve une décomposition analogue à celle faite plus haut entre “modèles du domaine” (ici, essentiellement la couche 1) et “modèles du raisonnement” (les couches 2, 3 et 4).

(iii) Convergence des approches

Au-delà d'un vocabulaire parfois disparate, ces différents travaux partagent une même motivation : appuyer la conception d'un système expert sur des *primitives d'un meilleur niveau d'abstraction*. Le terme “knowledge level” souvent employé pour désigner un tel niveau est dû à Newell (1981) qui l'oppose au “symbol level”. Ce “knowledge level” caractérise ce que fait le système et les connaissances utilisées, indépendamment du formalisme de représentation.

Les travaux décrits dans les sections précédentes proposent des primitives de modélisation que Steels (Steels 1990) a cherché à unifier

dans une même terminologie. Les trois concepts de base seront alors : les *tâches*, les *méthodes de résolution de problèmes* et les *connaissances du domaine*.

- *Les tâches*. Une tâche va être associée à un *but* à atteindre. Elle va être caractérisée par les propriétés des entrées, les résultats attendus en sortie et la nature des opérations permettant le processus de transformation. Une tâche de classification, par exemple, va indiquer en sortie les classes d'une taxinomie les plus représentatives d'une instance réelle particulière fournie en entrée.

- *Les méthodes de résolution de problèmes* (ou *Problem Solving Methods*). C'est la manière d'exécuter une tâche particulière, de satisfaire un but. Plusieurs méthodes peuvent s'appliquer à une même tâche. Par exemple, pour une même tâche de classification, on peut imaginer une grande variété de méthodes de résolution (Steels 1990) : recherche linéaire, recherche par raffinement descendant, recherche associative, différenciation, calculs de distances, etc. Une méthode particulière peut induire une décomposition de la tâche qu'elle résout en sous-tâches.

- *Les connaissances du domaine* (ou *Domain Knowledge*). Elles représentent l'univers du problème, et plus particulièrement la partie utile aux méthodes de résolution de problèmes (par exemple une taxinomie de pannes dans un système de classification)

Pour une discussion plus précise de ces concepts on se reportera à Steels (1990), Karbach & al. (1990) ou Vanwelkenhuysen & Rademakers (1990) par exemple. Tâches et méthodes de résolution de problèmes vont légèrement varier selon les auteurs. Clancey produira une des premières analyses détaillées d'une tâche particulière (classification). Chandrasekaran et son équipe chercheront à identifier un ensemble varié

de tâches pouvant servir de base à une *boîte à outils* (*Generic Task Toolset*). McDermott sera un des premiers à aborder le problème de plusieurs méthodes pour une même tâche. Au delà d'un vocabulaire pas toujours très homogène, de réelles convergences se dessinent aujourd'hui.

Nous avons montré que l'acquisition des connaissances nécessitait la *construction* d'un modèle du problème que l'on veut traiter. Cela revient à identifier les grandes tâches réalisées, pour chacune d'elles, spécifier les méthodes mises en oeuvre et donc les sous-tâches qui en découlent. Pour cela, il est nécessaire de disposer de primitives au bon niveau d'abstraction. Les travaux décrits plus haut traitent ce problème. Le modèle en 4 couches de KADS, les trois concepts proposés par Steels, les “role-limiting-methods” de McDermott sont relativement équivalents pour notre propos dans cet article. Ils constituent des *primitives* de modélisation. Dans la suite, nous appellerons *modèle conceptuel* un modèle construit à l'aide de ces primitives[2]. Sur cette base, nous indiquerons plus loin des outils d'aide à l'acquisition des connaissances qui ont été récemment développés.

3. Une architecture reflétant le modèle conceptuel

La question de l'implémentation du système est une question cruciale. Comment passer d'un modèle conceptuel à un système réel ? Comment ne pas “perdre” les avantages d'une conception au bon niveau d'abstraction? En effet, analyse, spécification et implémentation sont sujettes à être remises en cause au fur et à mesure de l'avancement d'un projet. On le conçoit alors, le modèle conceptuel doit se refléter sous une forme ou une autre dans l'*architecture* du système. Faute de quoi, toute la formalisation

---

[2] Une définition applicable à l'informatique classique est donnée dans (Aussenac 1989) et semble particulièrement adaptée : *“Le modèle conceptuel est un cadre sémantique partagé par des utilisateurs d'un programme et ses développeurs qui leur permet de communiquer”*. On trouvera également une définition intéressante dans (Norman 1983).

du problème décrite précédemment risque de ne s'avérer qu'une belle construction “pour l'esprit”. En outre, la qualité de l'architecture va avoir d'autres avantages que la seule aide à l'acquisition des connaissances : la maintenance et l'évolution de la base de connaissances seront facilitées, les explications générées seront de meilleure qualité.

Chandrasekaran propose une architecture adaptée à chacune des tâches génériques identifiées. Chaque tâche possédant ses propres connaissances et sa propre stratégie de résolution sera implémentée dans un formalisme adapté. La boîte à outils proposée consiste en un regroupement de “shells” de développement pour chacune des tâches génériques. Il y a d'ailleurs correspondance entre les “shells” et les tâches génériques : ces dernières sont considérées comme des blocs constitutifs de base, non décomposables. Construire un système consiste alors à identifier les grandes tâches réalisées et sélectionner les “shells” correspondants. Le contrôle de l'ensemble reste néanmoins encore assez mal défini. Il est censé “émerger” de l'interaction entre les différentes tâches, celles-ci communiquant par envoi de messages. De récents travaux ont cherché à traiter différemment cette question (Punch & Chandrasekaran 1990).

La méthodologie KADS souligne également la nécessité d'une architecture reflétant le modèle conceptuel. Il est même fait référence à un “*semi-isomorphisme”* entre les modèles issus de l'analyse (en termes de tâches) et l'implémentation. Dans la pratique, ceci devrait se faire au travers d'une succession de modèles conduisant à des spécifications de plus en plus précises.

La décomposition en termes de tâches, méthodes et modèles du domaine présentée plus haut a suscité un certain nombre d’autres travaux visant à la réalisation d'architectures permettant une représentation explicite et plus naturelle de ces concepts (Pierret-Goldbreich & Delouis 1990), (Vanwelkenhuysen & Rademakers 1990) ou (Tong & Tueni 1990) par exemple. Il est intéressant de noter ici la convergence avec les recherches sur les “Blackboards” (Nii 1989) ou sur des architectures à

vocation plus large telles que SOAR (Laird & al. 1987), en particulier pour l'expression du “contrôle” dans ces systèmes.

IV. L'acquisition des connaissances

L'acquisition des connaissances a été définie essentiellement comme un processus de modélisation. Des typologies et des primitives ont été proposées dans le paragraphe précédent. Nous allons maintenant montrer comment elles permettent d'aider ce travail de modélisation. Deux étapes distinctes doivent être auparavant distinguées : l'étape de construction du modèle, l'étape d'instanciation du modèle précédemment construit.

Des outils différents viseront à faciliter chacune de ces étapes. Ces outils seront décrits dans le paragraphe 5.

1. Construction du modèle

La construction de modèles va s'appuyer, d'une part, sur l'ensemble des primitives définies plus haut, d'autre part, sur l'existence d'une bibliothèque de modèles génériques. Cette partie du processus d'acquisition des connaissances s'en trouve ainsi facilitée :

- Il est en effet beaucoup plus simple de construire un modèle par *raffinements successifs* à partir de modèles génériques que de le créer ex-nihilo. C'est la démarche suggérée par la méthode KADS (sélectionner et raffiner des modèles génériques). C'est également la logique des *Generic Tasks* de Chandrasekaran. Bien sûr, ce n'est pas toujours possible. La bibliothèque de tâches génériques peut être insuffisante, le problème à traiter peut être atypique ou d'un nouveau type.

- Dans de tels cas, s'il reste une part d'empirisme, on gagne au moins à se poser de bonnes questions, ou tout du moins de meilleures

questions. Un modèle du type de celui proposé dans KADS (le modèle d'expertise en 4 couches) indique la structure de ce qu'il faut construire (ce modèle est d'ailleurs également utile pour raffiner des modèles génériques préexistants). S'interroger sur le système que l'on construit en termes de *tâches*, *méthodes de résolution de problèmes* et *modèles du domaine* est mieux adapté que de se poser des problèmes de *chaînage avant, arrière* ou *ensemble de conflits*.

2. Instancier le modèle conceptuel

Une fois le modèle conceptuel élaboré, il s'agit de le “remplir”, c'est-à-dire d'y inclure les connaissances nécessaires. Bien que cette deuxième étape puisse rétro-agir sur le modèle précédemment construit, elle sera souvent clairement dissociée à partir d'un certain état d'avancement. Cette phase d'instanciation de modèles a été plus largement approfondie par différentes équipes, et depuis plus longtemps. Les avantages sont donc mieux identifiés, les outils automatiques d'aide sont plus nombreux. Le point clé est la bonne identification du *rôle* joué par chacune des connaissances dans le processus de résolution de problème. On sait précisément ce qui doit être acquis et quel rôle chacune des entités de connaissance va jouer. Parmi les avantages de l'approche, nous pouvons relever un certain nombre de points (David & Krivine 1988, Marcus 1988) :

- On sait ce qu'il faut acquérir. *L'acquisition des connaissances est guidée par le modèle* que l'on cherche à remplir. Chaque tâche, chaque méthode de résolution de problème nécessite des connaissances précises, pour un rôle bien spécifié.

- Du côté de l'expert, une bonne définition et une bonne compréhension de la *fonction* des connaissances demandées en facilite l'expression.

- On dispose d'une sorte de *métrique* sur l'avancement du travail de constitution de la base de connaissances. On peut ainsi identifier facilement les endroits où il manque des connaissances. Ceci peut être mis à profit dans une phase de mise au point du système (simulation des méthodes manquantes lors de la validation incrémentale d'une base encore incomplète). Le système sait également identifier lui-même les méthodes où les connaissances sont manquantes. Il lui sera alors possible, dans une phase de résolution de problème, de rechercher des méthodes alternatives pour la même tâche ou, éventuellement, de demander à l'utilisateur. Ainsi, le système peut avoir une meilleure compréhension de ce qu'il sait faire et de ce qu'il ne sait pas (encore) faire.

Sur cette base, et pour des tâches particulières, des outils et des méthodes ont été largement développés. C'est ce que nous allons décrire dans le paragraphe suivant.

V. Aide à l'acquisition des connaissances

Les aides à l'acquisition de connaissances que nous allons décrire se situent bien sûr explicitement dans le paradigme décrit dans cet article. Nous n'avons donc pas la volonté d'être exhaustifs en matière d'outils d'acquisition, ni même d'être complets sur chacun des outils décrits. En particulier, nous ne décrivons pas les outils ou les méthodes visant à aider à l'instanciation d'un modèle conceptuel en s'appuyant sur des modèles “plus profonds” — voir par exemple (Reynaud 1989) ou (Charlet 1989) — ou des outils visant une autre partie plus “amont” dans l'acquisition des connaissances — méthode KOD (Vogel 88), MACAO (Aussenac 1989) par exemple.

Dans l'aide à l'acquisition des connaissances, nous distinguerons des *outils* (logiciels visant à aider le cogniticien à construire son système) et

des *méthodes de conception*, qui peuvent ne pas être associées à un logiciel particulier. Ces méthodes peuvent être complètes (toutes les phases de l'acquisition décrites dans cet article) ou plus spécifiques (protocoles particuliers).

Enfin, nous avons regroupé ces aides en deux classes selon la distinction faite dans le paragraphe 4 : les aides à l'instanciation d'un modèle particulier et les aides à la construction d'un modèle conceptuel.

1. Aide à l'instanciation de modèles

Nous allons décrire deux systèmes développés à Carnegy Mellon University par l'équipe de John McDermott et portant sur deux méthodes relativement génériques. Nous verrons ensuite sur un tout autre exemple comment il est possible de mettre en œuvre un protocole portant sur un modèle conceptuel particulier.

(i) MOLE

MOLE (Eshelman 1988) est un des premiers systèmes de ce type. Il est à la fois un “shell” de développement de systèmes experts et un outil d'acquisition des connaissances. MOLE suppose que la tâche à réaliser est une tâche de c*lassification* et met en oeuvre une méthode de résolution de problème de type “expliquer et différencier” (“cover-and-differentiate”). Cette méthode consiste à rechercher dans un premier temps des explications exhaustives et exclusives pour des faits observés, des symptômes par exemples[3]. Dans un deuxième temps, le système va rechercher la meilleure explication en éliminant celles jugées moins satisfaisantes. L'utilisation du système suppose que l'on s'est assuré au préalable que le problème correspondait bien à ce type de modélisation (que le *modèle conceptuel* de MOLE est adéquat). McDermott suggère un

[3] La même méthode porte le nom d'*abduction* chez Chandrasekaran.

certain nombre de critères afin de s'en assurer : possibilité d'identifier un ensemble d'observations (symptômes, signes) à expliquer ; pour chacun d'entre eux, possibilité d'énumérer statiquement différentes explications ; représentation de la stratégie de résolution sous la forme “expliquer” puis “discriminer”, etc.

Les connaissances à acquérir pour construire un système avec MOLE sont alors clairement définies par le rôle qu'elles auront à jouer. Il faudra acquérir la liste des *observations*, pour chacune d'entre elles, les explications possibles, et enfin, les connaissances visant à discriminer parmi plusieurs explications concurrentes.

MOLE propose alors un outil automatique visant à aider à l'acquisition de ces connaissances. Une fois les observations à expliquer entrées dans le système, MOLE va construire un réseau causal représentant les différentes explications possibles. Puis il va chercher à y incorporer les connaissances permettant d'éliminer une explication. Ce processus est itératif. MOLE est capable d'identifier les “faiblesses” de son réseau pour ensuite aider à raffiner les connaissances.

MOLE a été utilisé pour la construction de plusieurs systèmes dans le domaine du diagnostic, bien que la tâche réalisée (classification) ne soit pas propre au diagnostic.

(ii) SALT

SALT (Marcus 1988) est un système d'aide à l'acquisition des connaissances pour une tâche de conception sous contraintes. Il s'applique à des problèmes de conception ou d'ordonnancement (affectation de ressources). Les systèmes générés par SALT mettent en oeuvre une stratégie de résolution de problème de type “Proposer et réviser” *(“propose-and-revise”)*. Le système expert va commencer par construire un premier plan approximatif. Ceci consiste à affecter des valeurs à un certain nombre de paramètres. Il va ensuite vérifier qu'aucune contrainte

n'est violée par ces valeurs. Pour chaque violation de contrainte, SALT va chercher des modifications permettant d'arriver à une solution admissible.

Cette méthode de résolution de problèmes va délimiter les rôles que vont jouer les différents types de connaissances :

- *Proposer une solution*. C'est l'affectation de valeurs à un ensemble de paramètres. Des *procédures* vont indiquer comment calculer des valeurs pour chacun des paramètres. Un paramètre pourra accepter plusieurs méthodes. Une procédure est représentée sous forme de *frame* regroupant des informations indiquant le paramètre qualifié, des préconditions éventuelles, une description de la procédure en elle même (calcul, table, etc.) ainsi qu'une justification indiquant l'origine la méthode.

- *Identifier les contraintes*. Différentes informations vont décrire une contrainte : le paramètre contraint, la nature de la contrainte, une précondition, une justification, etc.

- *Proposer une amélioration.* Une *réparation* va décrire la manière de remédier à une violation de contrainte. Chaque *réparation* va être décrite par plusieurs attributs (paramètre qualifié, type de violation de contrainte, méthode d'amélioration, critères de préférence, etc.).

Les connaissances de base nécessaires à SALT pour générer une base de connaissances sont ainsi énumérées : *paramètres*, *procédure*, *contraintes* et *réparation*. Une liste d'*attributs* jouant un rôle spécifique bien identifié dans la résolution de problème vont décrire chacun de ces objets. Mais si cette description constitue les briques élémentaires du système, il s'agit ensuite d'organiser l'ensemble de manière cohérente. SALT va alors construire un *réseau de dépendance* exprimant les relations mutuelles entre ces différents concepts. Ce réseau est en quelque sorte la représentation interne de la compréhension de SALT du processus de

résolution de problème. Les noeuds du réseau vont être constitués de *paramètres* ou de *contraintes*. Les relations seront de trois types : des relations “contribue-à” pour exprimer qu'une *procédure* concluant sur un paramètre utilise un autre paramètre, des relations “contraint” pour exprimer le contenu d'une *contrainte* et des relations “suggère-révision-de” pour exprimer le contenu des *réparations*.

Les connaissances sont maintenant clairement identifiées, leur rôle est très précisement spécifié. C'est sur cette base que SALT peut alors aider à acquérir les connaissances. Mais SALT fait plus que de fournir une aide pour “remplir” les attributs des différents concepts énumérés. SALT est bien plus qu'un “éditeur de base de connaissances”. Il utilise sa représentation du processus (le réseau de dépendance) pour effectuer un certain nombre de vérifications et d'actions. Il va vérifier la *complétude* du réseau (identification de noeuds où manquent des relations particulières, des noeuds non contraints), sa “*compilabilité*” (possibilité d'assigner des valeurs à tous les paramètres), son “*unicité*” et sa *connexité*, l'absence de boucle, etc. Il aidera à formuler des modifications en cas de problème.

SALT a été utilisé pour générer deux systèmes experts : VT pour la conception d'ascenseurs et un système expert pour l'ordonnancement d'ateliers flexibles (Marcus & al. 1988).

(iii) Protocole dans le système DIVA

Nous allons maintenant présenter un protocole développé pour une des tâches composant le système DIVA.

DIVA est un système expert dont le modèle conceptuel est composé de trois grandes tâches : une tâche d'abstraction de données et deux tâches de classification (reconnaissance de situations et classification de défauts ; cf. David & Krivine 1989). L'acquisition des connaissances pour la tâche la plus importante (reconnaissance de situations) s'est appuyée sur la mise en place d'un véritable protocole. Cette tâche de classification met en œuvre une méthode de résolution “établir/raffiner”. Le but est de reconnaître dans

un ensemble de situations types celles qui rendent le mieux compte d'une situation réelle observée. Les situations sont organisées au travers d'une hiérarchie de situations types (prototypes) que le système va parcourir. Comme nous l'avons dit plus haut dans cet article, la méthode de résolution d'une tâche particulière induit la décomposition en sous-tâches. Plusieurs méthodes peuvent coexister pour une même tâche. Ici, la stratégie “établir et raffiner” va se concrétiser sous la forme de deux tâches : la *tâche établir* et la *tâche raffiner*. Cette décomposition se poursuivra jusqu'à des actions non décomposables. Une trentaine de tâches vont ainsi structurer l'ensemble du système.

Poursuivons en décomposant maintenant la *tâche établir*. La méthode de résolution de cette sous-tâche consiste dans un premier temps à acquérir les traits les plus importants pour la situation particulière que l'on cherche à établir. Ensuite, si aucune “valeur de rejet” n'a été obtenue et si suffisamment d'informations significatives ont été acquises, le système va chercher à évaluer la correspondance entre la situation réelle et la situation typique. Enfin, si le prototype est suffisamment reconnu, des *suggestions* seront faites quant à la poursuite du processus (quels sous-prototypes envisager, dans quel ordre). Le contrôle effectif du parcours sera pris en charge par la *tâche raffiner* sur la base de ces suggestions (David & Krivine 1989).

De cette stratégie découlent logiquement trois sous-tâches appelées *acquisition des traits caractéristiques*, *évaluation de la correspondance* et *évocation*. Chacune de ces tâches ou méthodes associées va nécessiter des connaissances spécifiques, pour un rôle clairement défini. Ainsi, par exemple, la première de ces trois tâches va demander l'énumération des traits caractéristiques de la situation et la seconde va demander les connaissances permettant de juger la correspondance.

Sur cette base, un protocole systématique a pu être mis en place, structuré autour de “fiches prototypes” (David & Krivine 1988). La transcription de ces fiches dans le système se fait quasi-automatiquement.

En bout de chaîne, le contenu de la base de connaissances pourra être restitué sous une forme comparable à celle des fiches prototypes. L'expert pourra s'y retrouver et procéder à des vérifications et de nouvelles modifications. Une importante base de connaissances a pu ainsi se constituer ces deux dernières années (Ricard & al. 1989)

Ce type de protocole illustre les deux facettes du modèle conceptuel. D'un côté, son expression dans des termes significatifs pour l'expert (situation typique, reconnaissance de situations), d'un autre côté, dans des termes plus génériques (classification heuristique, stratégie de recherche).

Une illustration de préoccupations analogues ayant conduit à la réalisation d'un outil automatique d'aide à la constitution de la base de connaissances peut être trouvée dans (Musen & al. 1987).

2. Aide à la construction de modèles

Nous allons maintenant décrire trois ensembles de travaux adressant l'aide à la construction de modèles. Ils se placent donc en amont des outils et méthodes de la section précédente.

(i) KADS

KADS (Breuker & Wielenga 1985, Wielenga & Breuker 1986, Breuker & Wielinga 1989, Hickman & al. 1989) a déjà été présenté dans cet article. Si son apport original concerne l'aide à la création de modèles, il importe néanmoins de souligner que la méthodologie couvre la totalité des phases de la réalisation d'un système.

Dans un premier temps, des techniques de conduite d'interviews et de verbalisation vont permettre d'accumuler informations et connaissances sur le domaine. Celles-ci vont ensuite être structurées et organisées au travers d'un modèle conceptuel. Celui-ci va décrire l'expertise selon les 4 couches exposées plus haut (domaine, inférences, tâches et stratégies). Un langage semi-formel va aider à cette description. Ce modèle conceptuel

peut soit être construit par sélection dans une *bibliothèque de modèles d'interprétation génériques* puis raffinement, soit par construction directe.

Ensuite, à partir du modèle conceptuel, une succession de transformations va permettre de spécifier l'implémentation *(design model).*

La structure en 4 couches (le *modèle d'expertise*) et le modèle générique sont les deux éléments essentiels de la méthode. KADS apparaît comme très flexible (Karbach & al. 1990). Néanmoins, la tâche de construction du modèle conceptuel proprement dite reste très délicate et largement basée sur l'expérience. Avant d'être une méthode formelle, KADS apparaît encore pour le moment comme une “philosophie” de conception.

(ii) Generic Tasks

Nous avons indiqué plus haut les motivations à la base de l'approche “Tâches Génériques” (Chandrasekaran 1987). On peut expliquer l'approche, avec du recul et dans le vocabulaire précédemment introduit comme suit :

- Il existe un certain nombre de “problem-solving methods” qui permettent de traiter une large classe de tâches. Ce qui les caractérise, c'est leur faible nombre, leur ubiquité et le fait qu'elles couvrent réellement une large classe de problèmes à travers différents domaines. Ces méthodes sont appelées “*Tâches Génériques”* (GT, avec le risque de confusion de vocabulaire qui en découle maintenant) auxquelles des connaissances spécifiques sont associées.
- Pour chaque GT identifiée va être défini un *langage* de haut niveau au moyen de la réalisation d'un *outil*. Les GT vont ainsi apparaître comme des bloc élémentaires (*building blocks*) permettant, par composition, la conception de systèmes complexes. Ceci a conduit à la réalisation d'une boîte à outils (*Generic Task Toolset*), sorte d'atelier de conception de systèmes à base de connaissances.

Les principales GT de cette boîte à outils répondent aux tâches suivantes :

- *classification hiérarchique :* CSRL (Bylander & Mittal 1986).
- *synthèse* (conception) par raffinement descendant de plans prédéfinis : DSPL (Brown & Chandrasekaran 1986).
- *abstraction de données :* IDABLE (Sticklen 1987).
- *matching d'hypothèses* : HYPER (Johnson & Josephson 86).
- *abduction* ou de l'assemblage d'hypothèses : PEIRCE (Punch & al. 1986).
- *raisonnement basé sur un modèle de fonctionnement* : FR (Sticklen & Chandrasekaran 1989).

Par rapport à KADS, cette boîte à outils offre l'avantage de proposer un moyen direct d'implémentation (il s'agit d'outils *effectivement* implémentés). Mais cette approche ne fournit pas une méthode explicite de conception, n'est pas à proprement parler un outil d'aide automatique à l'acquisition des connaissances. Rien n'est explicité sur la *manière* de sélectionner une ou plusieurs tâches génériques pour un problème donné (David 1988).

(iii) RIME

RIME (Soloway & al. 1987, Bachant 1988) apparaît comme une méthodologie de développement de systèmes à base de connaissances. En fait, initialement RIME a été conçu pour aider à l'évolution et la maintenance du système R1 (appelé également XCON) de Digital Equipment (RIME est l'acronyme de ***R1 Implicit Made Explicit***). Opérationnel depuis le début des années 80, R1 voit sa base de connaissances en constante évolution. Le taux de modification des règles est de l'ordre de 40% à 50% par an. Cette évolution n'est pas toujours facile à réaliser. Il faut parfois inclure de nouvelles tâches dans le système. Les connaissances de contrôle étant largement noyées dans les prémisses des différentes règles, il est parfois nécessaire de restructurer une grande

partie de la base de connaissances, ce que seuls des programmeurs maîtrisant bien R1 arrivent à mener à bien (Bachant & McDermott 1984). Indiquons enfin les difficultés dues aux incohérences entre les différents programmeurs et les différents styles de programmation. Le but de la méthodologie RIME est donc d'aider les ingénieurs de la connaissance à faire évoluer le système en leur proposant des primitives, à la fois de bas niveau (techniques de programmation, d'organisation et numérotage des règles), et de haut niveau (stratégies et contrôle). RIME va jouer en fait un rôle d'interface entre un niveau conceptuel adapté aux besoins des gestionnaires de la base de connaissances et le niveau de l'implémentation du système. Cet interface consiste en une sorte d'interprétation ou de traduction des concepts de bas niveau selon une *typologie* composée des quatre niveaux suivants :

- contrôle (sélection des "*problem solving methods*")
- focalisation du processus ("*focus of attention*")
- structure de représentation
- conventions de programmation.

Pour chaque niveau, une méthodologie adaptée et des points de repères ont été définis. Le niveau de contrôle correspond à l'identification des "Problem-Solving Methods". RIME connait actuellement 32 méthodes. C'est particulièrement à ce niveau que RIME prend tout son sens (expliciter l'implicite du contrôle de R1/XCON). En 1988 a commencé une réécriture de R1/XCON avec RIME et les premiers résultats semblent positifs, en particulier du point de vue de la maintenance et des capacités d'évolution du système.

Bien qu'initialement développé pour R1, RIME a une portée plus générale, comparable au modèle d'expertise de KADS. De fait, il a été appliqué à d'autres domaines, d'autres systèmes. Mais surtout, il est le point de départ de travaux en cours qui visent à développer des outils

automatiques d'aide à l'acquisition des connaissances incluant la partie sélection/construction du modèle et la partie instanciation.

VI. Conclusions

Nous avons montré dans cet article comment, en considérant le processus d'acquisition des connaissances comme un processus de *construction de modèles*, de nombreux travaux convergent aujourd'hui pour proposer une typologie et des primitives de modélisation adaptées. A partir de là, des outils d'aide à l'acquisition des connaissances et des méthodologies sont développées qui vont se répartir selon trois catégories :

- L'aide aux interviews d'experts (techniques de conduite d'entretiens et de verbalisation). Ces aspects n'ont pratiquement pas été évoqués dans le présent article. Nous avons juste souligné, que pour indispensables qu'elles soient, ces techniques ne peuvent prendre leur sens que si des cadres conceptuels sont proposés pour organiser et modéliser les connaissances.
- L'aide à la *construction* du modèle conceptuel, par *abstraction* (à partir des verbalisation d'experts ou des recueils de connaissances) et par *sélection* et *raffinement* de modèles génériques (RIME, KADS, Generic Toolset). Cette partie est de loin la moins maîtrisée et fait l'objet de nombreuses recherches : établissement de bibliothèques de tâches et de méthodes, guide pour identifier et sélectionner les méthodes adaptées, etc.
- L'aide à l'instanciation de modèles (c'est-à-dire au “remplissage” de ceux-ci par les connaissances du domaine).

Les futurs systèmes d'aide à l'acquisition des connaissances devront essayer de proposer une intégration harmonieuse de ces différentes étapes : verbalisations d'experts, constitution du modèle conceptuel,

sélection des outils permettant de “remplir” la base de connaissances, d'instancier le modèle.

Nous avons indiqué un autre aspect qu'il importe de souligner à nouveau. Le but d'un concepteur de système à bases de connaissances est de construire un *artefact* et non pas de simuler ou reproduire le raisonnement *réel* d'un expert. Ainsi, le “modèle conceptuel” dont nous avons parlé tout au long de l'article et que le concepteur du système se doit d'élaborer, n'est pas a priori un modèle réellement préexistant dans le cerveau humain (ce serait une vision très simpliste et réductrice de l'intelligence humaine). Ce modèle est une construction commune de l'expert et du cogniticien visant à fournir un langage d'expression des connaissances relativement “naturel” aux yeux de l'expert et “performant” aux yeux du cogniticien.

La validation et la maintenance de base de connaissances ont également été abordées par un certain nombre de travaux. Il est en effet plus simple de retrouver ce qui a conduit le système expert à un résultat jugé erroné lorsque le rôle joué par chacune des connaissances est clairement identifié au sein des différentes stratégies de résolution de problèmes.

Mais l'acquisition des connaissances n'est pas le seul attrait de cette approche. On peut également observer des retombées sur la qualité des systèmes conçus. Ainsi, par exemple, un certain nombre de travaux se sont focalisés sur les capacités explicatives de systèmes construits sur une représentation explicite de ce que fait le système — les tâches à réaliser — et de comment il le fait — au travers des méthodes de résolution de problèmes et des connaissances du domaine (voir par exemple Hasling & al. 1984, Marcus 1988, Chandrasekaran & al. 1989, David & Krivine 1990.

Remerciements

Nous tenons à remercier ici Nathalie Aussenac, Isabelle Delouis, Marc Linster, Chantal Massip et Philippe Mazas pour leur relecture et leurs commentaires sur une première version de cet article.

Jean-Paul KRIVINE
EDF - Direction des Etudes et Recherches
1, avenue du Général de Gaulle
92140 CLAMART

Jean-Marc DAVID
RENAULT
Service Systèmes Experts
860 Quai Stalingrad
92109 BOULOGNE-BILLANCOURT

Article reçu en septembre 1990
Version révisée en mars 1991

Références

Aussenac (1989) Nathalie Aussenac. *Conception d'une méthodologie et d'un outil d'acquisition de connaissances expertes*, Thèse de l'université Paul Sabatier de Toulouse, octobre 1989.

Bachant, McDermott (1984) Judith Bachant et John McDermott. "R1 revisited : Four years in the trenches", *The AI Magazine* 5(3), pp. 21-32, 1984.

Bachant (1988) Judith Bachant. "RIME : Preliminary Work Toward a Knowledge-Acquisition Tool" *Automating Knowledge Acquisition for Expert Systems*, édité par Sandra Marcus, Kluwer Academic Publishers, 1988.

Booch (1986) Grady Booch. "Object-Oriented Development", *IEEE Transaction on Software engineering*, 12(2), pp. 211-221, février 1986.

Boose, Nagai (1987) John H. Boose et Arthur T. Nagai. "Knowledge Acquisition Research : A summary of the AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop", *Cognitive Engineering in the Design of Human-Computer Interaction and Expert Systems*, édité par G. Salvendy, Elsevier Science Publisher, 1987.

Breuker, Wielenga (1985) Joost Breuker et Bob Wielenga "KADS : Structured Knowledge Acquisition for Expert Systems", *Actes des 5èmes journées internationales "Les systèmes experts et leurs applications"*, Avignon, France 1985,

Breuker, Wielenga (1989) Joost Breuker et Bob Wielenga "Models of Expertise in Knowledge Acquisition", *Topics in Expert System Design : methodologies and tools.* Guida & Tasso Eds, North Holland Publishing Company 1989.

Bronstown & al (1985) Lee Brownston, Robert Farell, Elaine Kant et Nancy Martin. "*Programming Expert Systems in OPS5"*, Addison-Wesley Publishing, 1985.

Brown, Chandrasekaran (1986) D. Brown et B. Chandrasekaran. "Knowledge and control for a mechanical design expert system", *IEEE-Computer*, 19(7), Juillet 1986.

Bylander, Mittal (1986) T. Bylander et S. Mittal. "CSRL: A Language for Classificatory Problem Solving and Uncertainty Handling", *The AI Magazine* 7(3), pp. 66-77, 1986.

Chandrasekaran (1983) B. Chandrasekaran. "Towards a taxinomy of problem solving types", *The AI Magazine* 4(1), pp. 9-17, 1983.

Chandrasekaran (1987) B. Chandrasekaran. "Towards a Functional Architecture for Intelligence Based on Generic Information Processing Tasks", *Tenth International Joint Conference on Artificial Intelligence (IJCAI-87)*, pp. 1183-1192, Milan, Italie, Août 1987.

Chandrasekaran & al (1989) B. Chandrasekaran, M.C. Tanner et J.R Josephson. "Explaining Control Strategies in Problem Solving", *IEEE-Expert,* Printemps 1989.

Charlet (1989) Jean Charlet. *LEZARD : Acquisition des Connaissances et Gestion de l'Incertitude dans un Système Expert de Seconde Génération*, Thèse d'université, Paris 1989.

Clancey (1985) W.J. Clancey. "Heuristic Classification", *Artificial Intelligence* 27(3), pp. 289-350, 1985.

Clancey (1986) W.J. Clancey. "From GUIDON to NEOMYCIN and HERACLES in Twenty Short Lessons : ORN Final Report 1979-1985", *The AI Magazine*, Août 1986.

Clancey (1989) William J. Clancey. "Viewing Knowledge Bases as Qualitative Models", *IEEE-Expert*, 1989.

David (1988) Jean-Marc David. "Functional Architectures and the Generic Task Approach", *The Knowledge Engineering Review*, Vol. 3, numéro 3, 1988.

David, Krivine (1988) Jean-Marc David et Jean-Paul Krivine. "Acquisition de Connaissances Expertes à Partir de Situations Types", *Actes des 8èmes journées internationales "Les systèmes experts et leurs applications"*, Avignon, France , Mai 1988.

David, Krivine (1989) Jean-Marc David et Jean-Paul Krivine. "Designing Knowledge-Based Systems within Functional Architecture : the DIVA Experiment", *Fifth IEEE Conference on Artificial Intelligence Applications (CAIA)*, Miami, Mars 1989.

David & al. (1989) Jean-Marc David, Jean-Paul Krivine et Reid Simmons. “Préface des actes de la conférence sur les Systèmes Experts de Seconde Génération”, *Actes des 9èmes journées internationales “Les systèmes experts et leurs applications”*, Avignon, France, 1989.

David, Krivine (1990) Jean-Marc David et Jean-Paul Krivine. “Explaining Reasoning from Knowledge Level Models”, *European Conference on Artificial Intelligence (ECAI-90)*, 1990.

Davis, Lenat (1982) Randall Davis et Douglas B. Lenat. *Knowledge Based Systems in Artificial Intelligence*, New York, McGraw Hill, 1982.

Eshelman (1988) Larry Eshelman. “MOLE : A Knowledge-Acquisition Tool for Cover-and-Differentiate Systems”, *Automating Knowledge Acquisition for Expert Systems*, édité par Sandra Marcus, Kluwer Academic Publishers, 1988.

Farenny (1987) Henri Farreny. *Les systèmes experts, principes et exemples*, Cepadues Editions, 1987.

Hasling & al (1984) D.W. Hasling, W.J. Clancey et G. Rennels. “Strategic explanations for a diagnostic consultation system”, *Int. Journal of Man-Machine Studies* 20, 1984.

Hayes-Roth & al (1983) Frederick Hayes-Roth, Donald A. Waterman et Douglas B. Lenat. “An Overview of Expert Systems”, *Building Expert Systems*, édité par Frederick Hayes-Roth, Donald A. Waterman et Douglas B. Lenat, Technoledge Series in Knowledge Engineering, 1983.

Hickman & al (1989) Frank R. Hickman, Jonathan L. Killin, Lise Land, Tim Mulhall, David Porter et Robert M. Taylor. *Analysis for Knowledge-Based Systems, a practical guide to the KADS methodology*, Ellis Horwood, 1989.

Hoffman (1989) Robert R. Hoffman. “A survey of methods for extracting the knowledge of experts”, *Sigart Newsletter*, Special Issue on Knowledge Acquisition (108), Avril 1989.

Johnson, Josephson (1986) T. R. Johnson et J.R. Josephson. *HYPER: The Hypothesis Matcher Tool*, Rapport interne Ohio State University.

Karbach & al (1990) Werner Karbarch, Marc Linster et Angi Voss. “Models of Problem-Solving : One label - One idea ?”, *Fourth European Knowledge Acquisition for Knowledge-Based Systems Workshop* (EKAW-90), Amsterdam, Pays-Bas, Juin 1990.

Laird & al. (1987) John E. Laird, Allen Newel et Paul S. Rosenbloom. “SOAR : An Architecture for General Intelligence”, *Artificial Intelligence* 33, pp. 1-64, 1987.

Marcus (1988) Sandra Marcus. “SALT : A Knowledge-Acquisition Tool for Propose-and-Revise Systems”, *Automating Knowledge Acquisition for Expert Systems*, édité par Sandra Marcus, Kluwer Academic Publishers, 1988.

Marcus & al (1988) Sandra Marcus, J. Stout et John McDermott. “VT : An expert elevator configurer that uses knowledge backtracking”, *The AI Magazine* 9(1), pp. 95-112, 1988.

McDermott (1988) John McDermott. “Preliminary Steps Toward a Taxinomy of Problem-Solving Methods”, *Automating Knowledge Acquisition for Expert Systems*, édité par Sandra Marcus, Kluwer Academic Publishers, 1988.

Meyer (1988) Bertrand Meyer. *Object-oriented Software Construction*, Prentice Hall, International Series in Computer Science, 1988.

Musen & al (1987) Mark A. Musen, L. M. Fagan, D.M. Combs, E.H. Shortliffe. “Use of a domain-model to drive an interactive knowledge-editing tool”, *International Journal of Man-Machine Studies* 26(1987), pp. 105-121.

Newell (1981) A. Newell. “The Knowledge Level”, *The AI Magazine* 2(2), pp. 1-20, 1981.

Newell, Simon (1972) A. Newell et H. Simon. *Human Problem Solving*, Englewood Cliffs, N.J., Prentice-Hall, 1972.

Nii (1989) H. Penny Nii. “Blackboard Systems”, *The Handbook of Artificial Intelligence*, Volume IV, chapitre XVI. Barr, Cohen et Feigenbaum eds, 1989.

Norman (1983) Donald H. Norman. “Some Observations on Mental Models”, Dans *Mental Models*, New Jersey : Hillsdale, Gentner D., 1983.

Pierret-Goldbreich, Delouis (1990) Christine Pierret-Goldbreich et Isabelle Delouis. “TASK : Task Architecture for the Structuration of Knowledge”, Conférence spécialisée sur les systèmes experts de deuxième génération, *Actes des 10èmes journées internationales “Les systèmes experts et leurs applications”*, Avignon, France 1990.

Porcheron (1990) Marc Porcheron. *Utilisation de méta-connaissances pour la compilation de règles de production*, Thèse de l'université Pierre et Marie Curie, Paris, 1990.

Punch & al. (1990) W. F. Punch, M.C. Tanner et J.R. Josephson. “Design Considerations for PEIRCE: A High Level Language for Hypothesis Assembly”, *Expert Systems in Government*, McLean Virginie USA, 1986.

Punch, Chandrasekaran (1990) William F. Punch III et B. Chandrasekaran. “An Investigation of the Roles of Problem-Solving Methods in Diagnosis”, Conférence spécialisée sur les systèmes experts de deuxième génération, *Actes des 10èmes journées internationales “Les systèmes experts et leurs applications”*, Avignon, France 1990.

Reynaud (1989) Chantal Reynaud. *ADELE : un outil d'aide à l'acquisition des connaissances basé sur des justifications,* Thèse de l'université Paris-XI, Orsay 1989.

Ricard & al (1989) Benoît Ricard, Roger Chevalier, Jean-Charles Bonnet et Jean-Pierre Tiarri. “Testing Diva, A Turbine-Generator Diagnosis Expert System”, *International Symposium AIPAC'89, Advanced Information Processing in Automatic Control*, Nancy France, 1989.

Soloway & al. (1987) Elliot Soloway, Judy Bachant et Keith Jensen. “Assessing the Maintainability of XCON-in-RIME : Coping with the Problems of a VERY Large Rule-Base”, *Conférence AAAI-87,* 1987.

Steels (1990) Luc Steels. “Components of Expertise”, *The AI Magazine* 11(2), 1990.

Sticklen (1987) Jon Sticklen. *MDX2 : An Integrated Medical Diagnostic System*, Ph.D. thesis. Ohio State University, 1987.

Sticklen (1989) Jon Sticklen. “Distributed abduction in MDX2”, *Artificial Intelligence in Scientific Computation : towards Second Generation Systems.* Editeurs : Hubert, Kulikowski, David et Krivine. J.C. Baltzer Publishing, 1989.

Sticklen, Chandrasekaran (1989) Jon Sticklen et B. Chandrasekaran. “Integrating Classification-Based Compiled Level Reasoning with Function-Based Deep Level Reasoning”, *Applied Artificial Intelligence*. Numéro spécial, 1989.

Tong, Tueni (1990) Xuejun Tong et Michel Tueni. “CARMEN: A Platform for Building Second Generation Expert Systems”, Conférence spécialisée sur les systèmes experts de deuxième génération, *Actes des 10èmes journées internationales “Les systèmes experts et leurs applications”*, Avignon, France 1990.

Vanwelkenhuysen, Rademakers (1990) Johan Vanwelkenhuysen et Philip Rademakers. “Mapping a Knowledge Level Analysis onto a Computational Framework”, *European Conference on Artificial Intelligence (ECAI-90)*, 1990.

Vogel (1988) Claude Vogel. *Génie Cognitif,* Paris, Masson, 1988.

Wielinga, Breuker (1986) B.J. Wielinga et J.A. Breuker. “Models of Expertise”, *European Conference of Artificial Intelligence (ECAI-86),* 1986.