THÈSE

présentée par

Mathias CHAILLOT

pour obtenir le grade de **DOCTEUR**
de l'**I**nstitut **N**ational **P**olytechnique de **G**renoble

(Arrêté ministériel du 30 mars 1992)

Spécialité: **Informatique**

Une Architecture de Contrôle Réactif pour la Résolution Coopérative de Problèmes

Date de Soutenance: 20 septembre 1993

Composition du jury:

Président:	Augustin	LUX
Rapporteurs:	Monique	THONNAT
	Raja	CHATILA
Examinateurs:	Elisabeth	CROCHON
	Jean-Louis	ERMINE
	François	RECHENMANN

Thèse préparée au sein du **LETI** (CEA – Technologies Avancées)

Remerciements

Je remercie Monsieur Augustin Lux, Professeur à l'ENSIMAG, qui m'a fait l'honneur de présider le jury de cette thèse.

J'adresse une vive reconnaissance à Madame Monique Thonnat, Directeur de recherches de l'INRIA à Sophia-Antipolis, et à Monsieur Raja Chatila, Chargé de recherches du CNRS au LAAS, pour avoir accepté de juger mon travail.

Monsieur Jean-Louis Ermine, Professeur à l'INSTN, m'a fait le plaisir d'être membre du jury, ce dont je le remercie.

Je remercie Madame Elisabeth Crochon, Ingénieur au LETI, pour la confiance et la liberté qu'elle m'a accordées en encadrant ce travail.

Merci à Monsieur Pierre Darier, chef du Service d'Électronique, de Techniques Informatiques et Architectures de m'avoir accueilli dans son laboratoire et de m'avoir donné les moyens d'effectuer ce travail de thèse dans de bonnes conditions.

Je remercie le CEA, qui m'a financé dans le cadre d'un contrat de formation par la recherche

Mes amitiés à tous les occupants du 40.22, ainsi qu'à tous les membres du projet SHERPA (Alexandro, Bruno, Cécile, Danièle, Jérôme, Jutta, Michèle, Nathalie, Nina, Olga, Oliviers, Pierre F., Pierre G., Shong).

Merci à Francis Robach, Patrice Uvietta et Pierre-Louis Bossart pour les relectures et les répétitions qui m'ont permi d'aboutir.

Je remercie Jérôme Euzenat pour le soutien amical mais exigeant qu'il m'a apporté.

Enfin, je remercie François Rechenmann, Directeur de recherches de l'INRIA Rhône Alpes, pour ..., mais surtout pour la rigueur intellectuelle qu'il sait transmettre à ses thésards.

Sommaire

Liste des Figures

Introduction

Ce travail aborde la problématique de la réactivité dans un environnement de résolution de problèmes ($\mathcal{ERP}$) afin de définir l'organisation d'une architecture de contrôle pour piloter des modules algorithmiques dans un environnement dynamique [Chai94].

Environnements de résolution de problèmes

Si les processeurs performants sont facilement disponibles, les bibliothèques de calcul qui permettent de les exploiter restent difficiles d'accès. Leur utilisation exige des connaissances dans le domaine d'application, la programmation des appels de fonctions et la consultation d'une documentation volumineuse. Par exemple, pour le CEA, l'importance de l'existant (des millions de lignes de codes écrites en Fortran !) justifie l'étude de solutions pour gérer et faciliter l'utilisation des ces bibliothèques de calcul, avec comme objectif à long terme, la capitalisation des connaissances qui s'y rapportent.

Plusieurs approches abordent le problème de la gestion des bibliothèques de calcul. Certains systèmes affranchissent l'utilisateur des contraintes de programmation au moyen de menus (ADECO en analyse de données [Ches89]) ou d'éditeurs graphiques de plans (Khoros [Will92]). D'autres systèmes généralisent cette approche en tentant d'automatiser l'utilisation de modules algorithmiques. Un environnement de résolution de problèmes ($\mathcal{ERP}$) est un système à base de connaissances qui aide à planifier et à contrôler l'exécution d'actions pour résoudre un problème. Générique ou lié à un domaine particulier, il comporte un ensemble d'actions, qui dans le cas du calcul scientifique, sont les modules algorithmiques d'une bibliothèque spécialisée. Les $\mathcal{ERP}$ abordent des problèmes plus ou moins complexes avec plus ou moins d'autonomie par rapport à l'utilisateur.

Environnements réactifs de résolution de problèmes

Nous nous intéressons plus particulièrement aux $\mathcal{ERP}$ réactifs, capables de mener à bien la résolution de problèmes de façon autonome dans un environnement dynamique. Ces systèmes ne coopèrent donc pas avec l'utilisateur, mais réagissent aux évolutions de l'environnement physique en prenant en compte les données issues de capteurs. Le travail que nous avons mené concerne plus particulièrement les domaines d'application suivants :

- En contrôle non destructif, le problème posé consiste à détecter et caractériser les défauts d'une pièce en temps limité. La décision fournie par le système détermine l'acceptation ou le rejet de la pièce. Les actions disponibles sont d'une part les traitements associés aux capteurs, et d'autre part les modules algorithmiques qui exploitent les données issues des capteurs.
- En détection sous-marine, le système dispose d'un intervalle de temps limité pour analyser les signaux issus du réseau de capteurs et pour décider de la présence et du type d'un sous-marin dans la zone contrôlée. Les actions disponibles sont les codes de calcul spécialisés en traitement du signal.

Le problème de la planification en résolution de problèmes a été largement traité, notamment en ce qui concerne la génération de plans. Notre système aborde des problèmes pour lesquels il existe des solutions. Ce sont des plans d'actions qui permettent de résoudre une partie du problème dans une situation bien déterminée. La problématique du contrôle dans un $\mathcal{ERP}$ réactif, telle que nous l'entendons, ne concerne donc pas la génération de plans mais plutôt la mise en œuvre de solutions existantes. Ces solutions sont nombreuses, et les situations auxquelles le système est confronté sont a priori très diversifiées et se succèdent de façon rapide par rapport au temps d'exécution d'une solution. Il est donc intéressant de disposer d'un système de décision qui contrôle la planification et l'exécution des solutions disponibles.

Problèmes abordés

Notre objectif est de concevoir un $\mathcal{ERP}$ réactif qui soit flexible et explicable. La flexibilité concerne les connaissances de contrôle et les modules algorithmiques qui doivent pouvoir être modifiés sans remettre en cause le système. L'explication exige que la résolution soit menée de façon explicite pour que l'activité interne du système soit accessible au module de production de l'explication. Les problèmes abordés dans ce travail sont liés d'une part aux spécificités des $\mathcal{ERP}$, et d'autre part à l'introduction des aspects temps réel dans un système à base de connaissances :

- La modélisation de l'état courant de la résolution doit détecter les événements qui sont pris en compte et reconnaître et caractériser les situations qui intéressent le contrôle.

- Le contrôle réactif doit être modélisé de façon explicite pour être couplé avec la gestion de la planification.
- Les temps d'exécution des mécanismes d'inférence doivent être maîtrisés pour respecter les contraintes de temps des réponses qui sont associées aux différents événements.
- La gestion des données issues des capteurs doit être compatible avec les inférences qui les utilisent.

La conception des $\mathcal{ERP}$ coopératifs constituent actuellement un important thème de recherche, de même que celle de systèmes à base de connaissances réactifs. Cependant, aucun $\mathcal{ERP}$ réactif n'a été proposé à notre connaissance. Il est donc nécessaire de s'intéresser aux différentes solutions qui interviennent dans la conception de ces deux types de systèmes.

Les $\mathcal{ERP}$ coopératifs mettent en œuvre différentes techniques pour planifier et contrôler l'exécution des modules algorithmiques. Parmi ces techniques, les tâches constituent la solution la mieux adaptée aux caractéristiques des $\mathcal{ERP}$.

Les systèmes à base de connaissances réactifs ou temps réel sont nombreux et abordent des domaines variés comme la résolution de problèmes, le contrôle d'un robot ou le contrôle d'un procédé industriel. Leur organisation varie selon que les couches de gestion de la réactivité et de la planification sont explicitement dissociées ou non.

Une architecture de contrôle réactif pour la résolution coopérative de problèmes

Nous avons réalisé une architecture de contrôle réactif, TRAM, qui reprend l'organisation en couches des systèmes à base de connaissances réactifs. La planification et le contrôle d'exécution des modules algorithmiques sont gérés à partir des notions de tâche et de méthode. Un superviseur défini par un automate d'états finis met en œuvre le contrôle réactif : en réponse à un événement, il sélectionne la tâche qui doit être réalisée.

Nous proposons dans ce travail l'approche des systèmes de production d'objets pour organiser un $\mathcal{ERP}$ réactif. Cette approche modélise l'état courant de la résolution dans une base d'objets qui met en œuvre deux mécanismes d'inférence : le mécanisme de classification d'instances caractérise les objets et le mécanisme de détection d'événements caractérise globalement la base. Les mises à jour de cette base sont effectuées par les capteurs et par les actions qui sont pilotées par le système. A partir de cette base d'objets, nous proposons un mécanisme de sélection dynamique de méthodes par sélection hiérarchique et gestion des possibilités d'activation.

Plan du mémoire

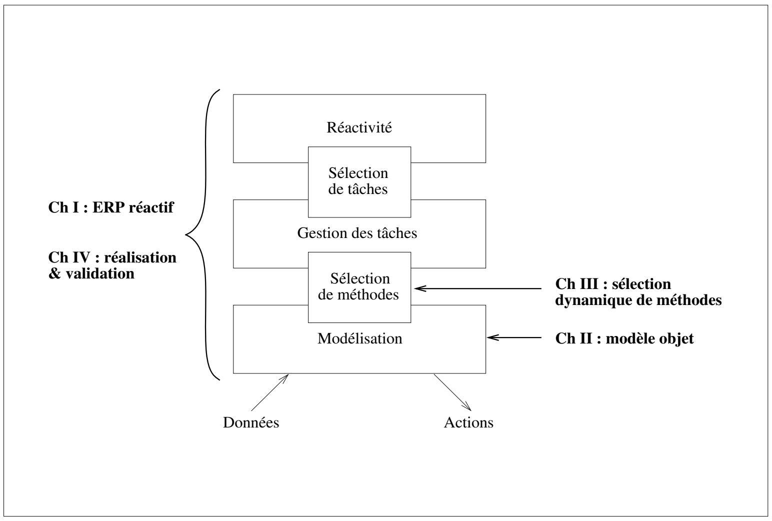

FIG. .1 - *Organisation de TRAM et distribution des chapitres*

Chapitre I : Environnement réactif de résolution de problèmes

Ce chapitre décrit l'organisation générale de notre architecture TRAM, un système à base de connaissances réactif qui pilote des modules algorithmiques. Il étudie tout d'abord la problématique associée à cet objectif. Puis il présente les différentes solutions retenues : le contrôle réactif est défini à partir d'un automate d'états finis; l'approche de la rationalité limitée est retenue comme approche des inférences temps réel. Enfin, l'architecture TRAM qui met en œuvre ces solutions est définie.

Chapitre II : Représentation des événements et caractérisation de la résolution

Ce chapitre définit de façon formelle le modèle objet qui est utilisé pour modéliser l'état courant de la résolution dans TRAM. Ce modèle objet combine les mécanismes d'inférence par classification d'instances et par détection d'événements. Des algorithmes d'évaluation de l'état courant de la base à partir de ses mises à jour sont proposés.

Chapitre III : Sélection dynamique de méthodes

Ce chapitre présente quatre approches pour la sélection dynamique de méthodes dans un système de représentation par objets des tâches et des méthodes. La première approche est celle de la sélection hiérarchique, elle consiste à classer une instance qui modélise l'état courant de la résolution vis-à-vis de la tâche à réaliser. En explicitant l'activabilité et l'utilité des méthodes associées à cette tâche, elle permet de sélectionner la méthode optimale. La deuxième approche adapte cette approche aux exigences d'un système réactif. Elle définit un mécanisme de classification rationnelle d'instances pour obtenir une sélection optimale en fonction du temps disponible. La troisième approche consiste à utiliser les événements gérés par la base pour détecter l'activabilité des méthodes. Enfin, une approche mixte est présentée, elle associe la sélection de méthodes par sélection hiérarchique avec la gestion des possiblités d'activation.

Chapitre IV : Réalisation et expérimentation

Ce chapitre décrit la réalisation du système TRAM sous la forme d'une architecture client/serveur. Les applications développées à partir de TRAM sont ensuite présentées. En perspective, la notion d'hypertrace est proposée pour expliquer et évaluer la résolution : une hypertrace est une trace de type hypertexte qui fournit une description de haut niveau de l'activité interne du système.

Chapitre I

Environnement réactif de résolution de problèmes

Ce chapitre décrit l'organisation générale de notre architecture TRAM, un système à base de connaissances réactif qui pilote des modules algorithmiques.

Nous étudions tout d'abord la problématique associée à cet objectif (I.1). Puis nous présentons les différentes solutions retenues : le contrôle réactif est défini à partir d'un automate d'états finis (I.2); comme approche des inférences temps réel nous privilégions celle de la rationalité limitée (I.3). Enfin nous définissons l'architecture TRAM qui met en œuvre ces différents choix (I.5).

I.1 Problématique

Les systèmes à base de connaissances qui ont pour fonction de résoudre des problèmes en pilotant les modules algorithmiques d'une bibliothèque de calcul sont appelés des environnements de résolution de problèmes ($\mathcal{ERP}$). Notre objectif est donc de concevoir un $\mathcal{ERP}$réactif.

I.1.1 Les environnements de résolution de problèmes

Les $\mathcal{ERP}$ constituent un domaine dont l'activité de recherche est motivée par l'objectif de mettre les bibliothèques de codes de calcul à la portée du plus grand nombre d'utilisateurs possible. Alors que la disposition de micro-processeurs performants est banalisée, les bibliothèques d'informatique scientifique ou technique exigent des connaissances pointues dans le domaine concerné et restent complexes à utiliser tant du point de vue de la programmation que du point de vue de la documentation à consulter. Le poids de l'existant étant ce qu'il est, cette situation n'est pas appelée à évoluer rapidement. Dès lors, plusieurs approches sont actuellement envisagées pour faciliter l'utilisation de ces bibliothèques.

Définitions

"Un environnement de résolution de problèmes est un système informatique qui fournit des services pour résoudre une classe de problèmes". Cette définition peu précise a été proposée dans une conférence sur les $\mathcal{ERP}$ [Gall91]. Elle recouvre un ensemble vaste de logiciels qui se caractérisent par les services qu'ils offrent et par leurs domaines d'application.

Le premier objectif visé par ces systèmes est de libérer l'utilisateur des contraintes de programmation. Le système ADECO [Ches89] permet de combiner des séquences de traitements en analyse de données. A partir de menus, il fournit une documentation en ligne et aide à réaliser les analyses multivariées dans le domaine de l'écologie. Les langages de programmation visuelle proposent un éditeur de plans d'actions qui permet de spécifier des séquences de traitement en positionnant des boîtes qui représentent les traitements. Les modules *Cantata* et *ICE* sont les éditeurs de plans respectivement des systèmes *Khoros* [Will92] et *VIDIMUS* [1]. Outre la définition des plans, ils génèrent et exécutent les programmes correspondants. *Khoros* a pour rôle d'organiser les fichiers sources pour constituer une bibliothèque de calcul. Ces services diminuent la charge de programmation pour éliminer les aspects fastidieux liés à l'utilisation d'une bibliothèque de calcul, mais ils n'abordent pas le problème des connaissances requises : l'utilisateur a la charge de sélectionner les traitements dont il a besoin.

L'étape suivante dans l'aide à l'utilisation de bibliothèques de calcul consiste à piloter les modules algorithmiques. Cet objectif ambitieux motive de nombreux travaux de recherche, tels que le projet *VIDIMUS* qui vise à automatiser le traitement d'images. Dans ce travail, nous nous intéressons aux $\mathcal{ERP}$ qui permettent de piloter des modules algorithmiques, et parmi ceux-ci, à ceux qui constituent des systèmes à base de connaissances.

Un système à base de connaissances est un système informatique qui comporte d'une part une modélisation des connaissances relatives à la résolution d'une classe de problèmes et d'autre part une représentation des processus d'inférence qui opèrent sur cette modélisation pour résoudre un problème donné [Chan92]. L'opposition classique entre les approches procédurale et déclarative est insuffisante car les langages réactifs [Dzie90] sont déclaratifs mais ne défi-

[1] Projet ESPRIT II 2592 VIDIMUS

nissent pas des systèmes à base de connaissances. C'est donc la modélisation explicite du contrôle qui caractérise les systèmes à base de connaissances par rapport aux systèmes de production de règles qui appliquent un algorithme pour exploiter les connaissances modélisées sous la forme de règles. *La capacité de modélisation d'un système à base de connaissances est sa capacité à aborder une large catégorie de problèmes [Doyl91].*

Systèmes à base de connaissances et explication Rappelons que la mise en œuvre d'un système à base de connaissances est motivée par la représentation explicite des connaissances de résolution. Si l'interprétation de cette représentation aboutit à une dégradation des performances, la modification aisée de cette représentation rend le système évolutif et surtout la représentation explicite de l'activité interne du système pendant la résolution permet d'aborder l'explication (voir IV.4). Les systèmes à base de connaissances sont donc indissociables de la mise en œuvre de l'explication et nous porterons un intérêt tout particulier à cet aspect lors de l'argumentation de nos choix.

Planification et résolution de problèmes *Etant donnés une représentation de l'état initial du monde, un ensemble d'opérateurs de changement d'états et un but à atteindre (le problème), la planification consiste à trouver une suite d'actions (plan) à appliquer pour passer de l'état initial à un état qui satisfait le but à atteindre.* Les actions sont des exécutions d'opérateurs. Il y a résolution de problème s'il existe un moyen pour évaluer les solutions proposées. Les données de la résolution sont celles qui sont manipulées par les opérateurs. La planification est historiquement liée à la robotique, une synthèse est donnée dans [Regn90]. La génération de plans concerne la production de plans sans disposer de solutions prédéfinies.

Parmi les systèmes à base de connaissances pour la résolution de problèmes, les $\mathcal{ERP}$ constituent un sous-groupe de systèmes dont les spécificités sont liées à leurs objectifs et aux services qu'ils doivent fournir plutôt qu'à une définition particulière.

Nous donnons la définition suivante : *Un environnement de résolution de problèmes est un système à base de connaissances qui aide l'utilisateur à planifier et à contrôler l'exécution d'actions pour résoudre une catégorie de problèmes.* Dans le contexte du calcul scientifique ou technique, les actions qui sont pilotées par le système sont les modules algorithmiques d'une bibliothèque logicielle spécialisée dans un domaine comme le traitement du signal et l'analyse de données.

Objectifs des $\mathcal{ERP}$

Trois objectifs caractérisent les $\mathcal{ERP}$:

1. La **modélisation des connaissances** relatives à la résolution d'une catégorie de problèmes dans un domaine donné constitue en soi un objectif

difficile à atteindre et intéressant pour les spécialistes de ce domaine. Cet objectif vise à capitaliser les connaissances liées à la résolution. Même dans les domaines aussi bien formalisés que la résolution d'équations aux dérivées partielles [Chai89b], le calcul de valeurs et de vecteurs propres, l'analyse de données [Chev91], cette modélisation fait l'objet d'un travail à part entière (et souvent d'une thèse!).

2. Si elle est opérationnelle, cette modélisation peut être mise à la disposition des utilisateurs pour résoudre des problèmes. Le système est alors capable de **piloter une bibliothèque de modules algorithmiques** dont l'utilisation peut exiger une documentation et des connaissances complexes.

3. A partir de la modélisation des connaissances et de la trace de la résolution, **l'explication** et **l'évaluation** de la résolution sont possibles. Elles peuvent être réalisées dans le but de former l'utilisateur, de vérifier ou d'analyser le comportement du système.

Fonctionnalités des $\mathcal{ERP}$

Les fonctionnalités des $\mathcal{ERP}$ sont multiples [Gall91] : parmi les services qu'ils peuvent fournir à l'utilisateur, figurent les éléments suivants qui relèvent de domaines variés :

- Un état de l'art des solutions possibles dans le domaine concerné.
- Une sélection plus ou moins automatique des modules algorithmiques qui réalisent ces solutions.
- Des facilités pour modifier les modules algorithmiques ou pour en intégrer de nouveaux.
- Une gestion des ressources disponibles pour l'exécution des modules algorithmiques en parallèle ou en séquence.
- La construction d'une trace qui permet d'analyser et d'évaluer la résolution.
- Des possibilités de reprise des calculs par modification de certains paramètres.
- Une interface de haut niveau qui fournit une représentation graphique des plans d'actions et qui permet de visualiser les données.

Tous ces services ne sont pas disponibles dans les différents $\mathcal{ERP}$ existants, mais ils font partie des évolutions actuelles [Hous93].

Organisation générale des $\mathcal{ERP}$

Le rôle des $\mathcal{ERP}$ qui consiste à piloter des systèmes logiciels externes influence leur organisation en tant que système d'infomation. A. Newell a utilisé le système Soar [Newe93] pour réaliser un $\mathcal{ERP}$, et à partir de ses travaux sur la modélisation des connaissances, il a identifié l'organisation générale des $\mathcal{ERP}$ (Figure I.1) dont les composants sont les suivants :

1. une base de données qui contient la modélisation des entités du domaine.
2. une interface de communication qui gère l'exécution des actions.
3. un module de contrôle qui à partir du contenu de la base décide des actions à déclencher.

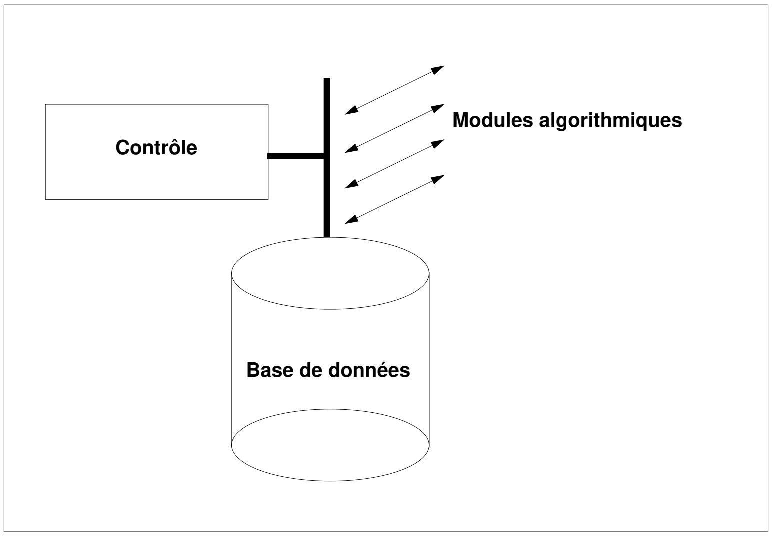

FIG. I.1 - *Organisation générale des $\mathcal{ERP}$ d'après [Newe93]*

.

En fait, ce schéma d'organisation identifie seulement les composants. L'architecture du système peut varier selon la nature de la base de données, le type de contrôle et la localisation des modules algorithmiques.

Connaissances utilisées pendant la résolution

La modélisation des connaissances est le point le plus important. C'est elle qui conditionne la crédibilité du système en lui permettant d'aborder des problèmes non triviaux dans les domaines concernés. Les connaissances de résolution qui doivent être modélisées dans un $\mathcal{ERP}$ concernent les éléments suivants :

1. les entités du domaine,

2. les problèmes traités,

3. les actions (modules algorithmiques) disponibles,

4. le contrôle (les principes de résolution : enchaînements d'actions, réactions aux événements) qui spécifient l'utilisation des actions disponibles pour mener à bien la résolution des problèmes.

Il est donc clair que l'intérêt d'un $\mathcal{ERP}$ est directement lié à la modélisation des connaissances de résolution qu'il permet (a) d'expliciter, (b) de mettre en œuvre et (c) d'expliquer. C'est donc sur la puissance de cette modélisation que se concentrent les efforts de recherche. La modélisation des connaissances de contrôle ("Quoi faire ?" et "Comment le faire ?") est actuellement abordée au travers des notions de tâches et de méthodes. Le rôle des tâches et des méthodes est étudié dans le chapitre III.

I.1.2 Problèmes de la réactivité dans les $\mathcal{ERP}$

Un $\mathcal{ERP}$ réactif, au sens où nous l'entendons, est un système à base de connaissances pour la résolution de problèmes et non pour la conduite de procédés. Les $\mathcal{ERP}$ réactifs ne constituent donc pas des systèmes temps réel, car ils ne vérifient pas la définition donnée dans [Grou88]. Cependant, ils sont concernés par les problèmes liés à l'introduction des aspects temps réel dans un système à base de connaissances :

- **Modélisation de l'état de la résolution.** En résolution de problèmes, la modélisation de l'état de la résolution fournit les informations qui concernent l'état de l'environnement du système et les solutions qui sont construites. Dans un système réactif, cette modélisation doit permettre de détecter les événements auxquels le système doit réagir. Nous proposons dans le chapitre II un modèle objet qui permet de modéliser l'état courant de la résolution dans un système réactif dont l'objectif est de piloter des codes de calcul.

- **Planification.** La planification des actions a effectuer pour atteindre les buts courants est concernée par la réactivité dans la mesure où le temps qu'elle consomme doit être maîtrisé. Pour mettre en œuvre la planification, nous développons dans le chapitre III une approche basée sur les tâches et les méthodes.

- **Réactivité.** Pour que la réactivité soit réellement intégrée dans la définition d'un système à base de connaissances, il faut qu'elle soit modélisée au niveau du contrôle. Nous décrivons dans la section I.2 les solutions qui ont été proposées pour modéliser la réactivité dans un système à base de connaissances.

- **Maîtrise des mécanismes d'inférence.** Dans un système à base de connaissances, les algorithmes utilisés pour les inférences présentent des caractéristiques qui les rendent non prédictibles, donc incompatibles avec les exigences d'un système temps réel. Nous explicitons dans la section I.3 les différentes approches qui sont mises en œuvre pour contrôler les mécanismes d'inférence dans un système à base de connaissances réactif.

- **Gestion asynchrone des événements.** Un système à base de connaissances réactif met en œuvre les techniques du temps réel, en particulier pour prendre en compte les données issues des capteurs. Cependant, le fait que ces données doivent être intégrées dans la modélisation de la résolution soulève de nouveaux problèmes. Nous présentons donc en section I.4 les travaux qui ont été menés pour gérer les données issues de l'environnement extérieur dans un système à base de connaissances réactif.

I.2 Contrôle réactif en résolution de problèmes

La gestion de la réactivité en résolution de problèmes consiste à réagir aux événements pour identifier les actions qui constituent les réponses appropriées du système à ces événements.

I.2.1 Problème

Le *comportement* d'un système informatique est réactif s'il réagit à des événements en modifiant son état et en apportant des réponses à ces événements. La réponse à un événement donné est pertinente si elle intervient dans un délai déterminé.

"La *définition* d'un système informatique est réactive si elle spécifie son comportement et non les algorithmes de traitement des données" [Dzie90].

Un $\mathcal{ERP}$ est donc réactif si la résolution est en partie définie de façon réactive. Un problème lui est confié sous la forme d'une ou plusieurs tâches à réaliser. Il adapte la résolution du problème aux variations de l'environnement en modifiant son état interne et ne requiert pas l'intervention de l'utilisateur pour désigner la tâche principale à réaliser. La réactivité dans les systèmes de résolution de problèmes a été introduite pour faire face à la complexité des problèmes de planification. Parce que la complétude d'un système de planification ne peut pas être assurée pour des problèmes complexes, l'introduction de la réactivité permet de réduire cette complexité. L'approche réactive de la définition d'un système est organisée à partir des notions d'états et de transitions. En limitant la portée de la planification à une classe d'états, la complexité du problème de planification se trouve réduite [Curr91].

I.2.2 Approche des langages réactifs

Une approche possible est celle des langages réactifs (ELECTRE, ESTEREL, LUSTRE, SIGNAL [Dzie90]). Ces langages sont déclaratifs dans le sens où ils définissent le comportement du système réactif par opposition aux langages de programmation qui décrivent les algorithmes de traitement des données. Ils présentent deux caractéristiques intéressantes : ils abordent des problèmes du temps réel dur avec des contraintes de temps de l'ordre de la milliseconde; et ils admettent chacun un modèle mathématique qui permet de prouver le fonctionnement correct du système réactif défini à partir du langage. Cependant, la garantie d'une preuve formelle est obtenue au détriment de la complexité des représentations autorisées dans ces langages. Ils gèrent des représentations de bas niveau et leur associent des opérations élémentaires. Cette contrainte est bien adaptée aux systèmes temps réel : *"Un système temps réel est un système informatique dont le fonctionnement est assujetti à l'évolution d'un procédé dont il doit contrôler le comportement"* [Grou88]. Les langages réactifs concernent donc des applications où les représentations de données sont simples et où les traitements sont élémentaires, mais où les contraintes de temps sont fortes et les événements sont multiples.

Ces contraintes ne correspondent pas à celles d'un $\mathcal{ERP}$ réactif qui doit seulement réagir aux évolutions de l'environnement extérieur pour adapter la résolution du problème à ces événements et non assurer le fonctionnement correct d'un procédé industriel. Aussi les contraintes de temps sont-elles moins fortes et les occurrences d'événements moins nombreuses. Par contre, la modélisation des connaissances relatives à la résolution de problèmes est importante et, pour être opérationnelle sur des problèmes non triviaux, elle doit mettre en œuvre des représentations de données complexes et des traitements variés. L'approche des langages réactifs est donc insuffisante pour la définition des $\mathcal{ERP}$ réactifs, nous lui préférons celle des systèmes à base de connaissances.

I.2.3 Organisation des systèmes à base de connaissances réactifs

Une solution pour l'organisation des systèmes à base de connaissances réactifs distingue trois couches qui concernent respectivement le comportement réactif, la planification et la modélisation de l'état de la résolution. Les caractéristiques communes sont les suivantes :

1. Le choix du but à atteindre ("Quoi faire?") est à la charge de la couche réactive qui comprend la gestion des connaissances relatives au comportement du système. La solution commune aux différents travaux est basée sur une discrétisation de l'espace des états possibles du système. A partir de l'ensemble fini des états identifiés, son comportement est spécifié par la description des transitions qui associent aux événements pris en compte les changements d'états qui leur correspondent. Cette approche est identique à celle des systèmes temps réel [Bapt92].

2. L'utilisation de plans prédéfinis permet de rester efficace dans l'élaboration d'une réponse aux événements en ne déclenchant pas de mécanisme d'inférence pour construire un plan quand une solution connue existe. Les différentes variantes de l'utilisation de plans prédéfinis permet de résoudre le problème du "Comment faire?" lorsque la tâche principale à réaliser est fixée.

3. Pour modéliser les entités du domaine et de la résolution, pour identifier l'état reconnu du système, pour détecter les événements et pour gérer les tâches, ces systèmes utilisent une modélisation de l'état courant de la résolution.

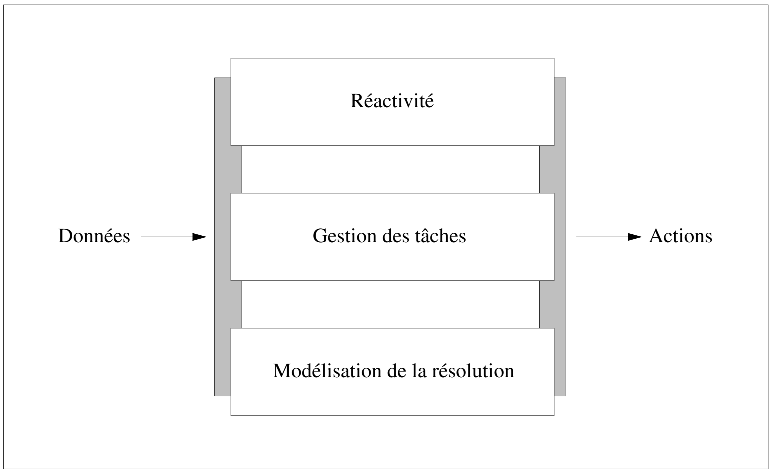

FIG. I.2 - *Organisation en couches des systèmes à base de connaissances réactifs.*

Les différences entre les systèmes à base de connaissances réactifs concernent la représentation et la gestion des plans ainsi que la représentation des connaissances relatives aux aspects réactifs. La modélisation de l'état de la résolution joue un rôle particulier dans les systèmes de type "blackboard", car elle doit permettre de gérer les possibilités d'activation. Les architectures "blackboard" sont présentées dans la section I.2.7.

I.2.4 Modélisation d'un environnement dynamique

L'objectif des premiers travaux liés aux systèmes à base de connaissances en environnement dynamique était de modéliser cet environnement. Ils reposent sur le postulat que le monde concerné est à chaque instant dans un état donné. De cet état sont connues des propriétés qui intéressent le système à base de connaissances. Un tel état n'est pas défini par sa durée mais par des occurrences d'événements. Ce sont les transitions qui définissent les changements d'états. Les formalismes associés à cette approche décrivent les événements pour définir les états de l'environnement et leurs propriétés (*calcul des situations* [McCa69]). La représentation des événements pour une application réelle soulève deux problèmes classiques : la gestion des propriétés valides pour un état après l'occurrence d'un événement (*Frame problem*) et la multiplicité des effets d'un événement (*Ramification problem*). Ces problèmes sont dus à la complexité d'un environnement dynamique dans le cas d'une application réelle.

La génération de plans dans un environnement dynamique est concernée par ces aspects. De nombreux travaux ont été consacrés à cette problématique. La logique temporelle de D. McDermott [McDe82] permet la planification d'actions qui sont des séquences d'états. Le calcul d'intervalles de J. Allen [Alle84] spécifie les actions à partir des relations entre les intervalles de temps que concernent leurs conditions et leurs effets. L'utilisation de ces formalismes pour des applications réelles reste limitée par la complexité des problèmes abordés.

La conception de systèmes à base de connaissances réactifs ou temps réel fait l'objet de nombreux recherches et développements, en particulier dans le cadre du projet de copilote (*Pilot's Associate*) [Lark90]. Nous ne présentons donc ci-dessous qu'une partie de ces travaux.

I.2.5 Contrôle non délibératif

Parmi les approches utilisées pour le contrôle réactif, le contrôle non délibératif met en œuvre des représentations de données sans inférence et dont la complexité est limitée pour être plus proche de la perception de l'environnement. Le comportement des systèmes basés sur le contrôle non délibératif est fortement réactif (on parle d'ailleurs d'architectures réactives), il est lié aux caractéristiques de l'environnement pour lequel il a été défini. La forte réactivité et la robustesse sont obtenues au détriment d'une représentation interne de l'activité du système, ce qui la rend donc incompatible avec l'explication. Cette approche est bien adaptée aux problèmes de contrôle de bas niveau mais est incompatible avec le niveau de modélisation requis pour un $\mathcal{ERP}$. Nous ne la citons donc que pour son caractère fortement réactif.

Brooks R. Brooks [Broo91] a développé une architecture de *subsumption*, pour sélectionner et exécuter des actions dans le contexte de la robotique. Plusieurs *modules de compétence* recoivent directement les données issues des capteurs et non une modélisation de l'environnement. Chaque module de compétence fonctionne indépendamment des autres selon un automate d'états finis et traite ses entrées tant qu'il n'est pas inhibé (*subsumed*) par un autre module. Cette architecture constitue une réalisation de mécanismes de calcul et non pas une architecture de sélection de méthodes.

Maes P. Maes [Maes90] a proposé un algorithme de sélection d'actions. Dans son modèle, un agent dispose d'un ensemble de *modules de compétences* dont la granularité est supérieure à celle des neurones en approche connexioniste. La définition de chaque module comporte sa condition d'activation, ses effets, son niveau d'action et ses interactions avec les autres modules. Les modules sont associés aux nœuds d'un réseau, et une mesure de perception de chaque nœud est calculée en fonction de l'état des nœuds en relation avec lui. Les entrées du réseau sont fournies par des capteurs et par les buts de l'agent. La sélection d'actions consiste alors à fixer un seuil d'activabilité, à rechercher parmi les actions activables celle qui a la plus forte activabilité et à l'exécuter.

I.2.6 Contrôle délibératif

Alors qu'une architecture non délibérative associe directement aux événements les réponses pertinentes, une architecture délibérative construit l'ensemble des possibilités et met en œuvre un mécanisme de prise de décision pour sélectionner l'une d'entre elles [Bach92]. Les architectures délibératives abordent des problèmes de plus haut niveau que les systèmes réactifs.

La présentation de systèmes réactifs est destinée à illustrer comment ils modélisent et gèrent les connaissances relatives à la réactivité. La modélisation du domaine n'est donc pas abordée.

Systèmes

Plusieurs architectures délibératives ont été développées pour le contrôle réactif. Nous présentons ci dessous les principaux, hormis les systèmes "blackboard" qui sont étudiés plus particulièrement en I.2.7.

M. Georgeff - PRS (Procedural Reasoning System) est une architecture de contrôle réactif développée pour piloter un robot autonome mobile [Geor89]. Elle est constituée (a) d'une base de données qui contient la modélisation de la résolution, (b) d'un ensemble de buts à atteindre, (c) d'une librairie de procédures et de plans qui décrivent des séquences d'actions et de tests pour réaliser certains buts et pour réagir aux situations et (d) d'une structure d'intentions sous la forme d'un ensemble partiellement ordonné de plans [Ingr92].

Un interpréteur sélectionne un plan à partir de l'état de la résolution et des buts actifs et le place dans la structure d'intention. Les mécanismes de gestion de cette structure utilisent une priorité sur ces plans pour exécuter son contenu.

PRS interagit avec l'environnement par l'intermédiaire de sa base et par les actions qu'il déclenche pour réaliser ses intentions. La représentation explicite de ses buts (*KA : Knowledge Area*) prend la forme d'un graphe qui peut être vu comme un plan ou un schéma de plan. Chaque arc est étiqueté par le but à atteindre. La condition d'invocation de chacun de ces éléments est structurée par une partie déclenchante qui décrit logiquement l'événement qui doit intervenir pour l'exécuter et par une partie contextuelle qui spécifie les conditions qui doivent être vérifiées.

Le cycle d'inférence est organisé de la façon suivante : - les buts sont fixés et des événements surviennent - un ou plusieurs KA deviennent déclenchables - un ou plusieurs KA sont sélectionnés et placés dans la structure d'intention - une tâche (intention) est exécutée : elle peut soit exécuter une action physique, soit établir un nouveau but, soit modifier l'état de la base pour changer la modélisation de la résolution.

P. Cohen - Phoenix est un environnement multi-agents expérimenté dans le cadre de l'extinction d'incendies de forêts [Howe90]. Son objectif est de permettre une étude approfondie des agents réactifs en fonction de leur environnement. Chaque agent est constitué de deux composants : un module réflexe associé à un composant de planification qui assure la communication avec les autres agents. Les deux composants recoivent des informations par les capteurs et peuvent exécuter des commandes.

Le module réflexe associe aux événements détectés par les capteurs des fonctions qui déterminent les actions à effectuer pour leur apporter une réponse. Le rôle de ce module est d'assurer une réactivité maximale aux événements critiques en évitant de les prendre en compte dans un système de représentation complexe destiné à la planification. Ces réflexes peuvent interrompre le plan qui était en cours d'exécution, le module d'exécution avertit alors le module de planification de cette interruption.

Le module de planification met en œuvre la méthode d'expansion mesurée de squelettes de plans (*Lazy Plan Expansion*) qui consiste à instancier des plans prédéfinis et à adapter cette instance aux contraintes de temps imposées par l'environnement. Il gère également les interruptions qui sont issues du module réflexe en associant à chaque sous-ensemble d'un plan une description des traitements d'erreur et de reprise.

M. Musen - ONCOCIN est une architecture de contrôle réactif dédiée au suivi de malades [Tu89]. Elle étend la méthode des raffinements de plans pour prendre en compte les événements du passé et la durée des actions. Elle est organisée à partir des modules suivants : (a) une base de connaissances qui comporte les squelettes de plans et les heuristiques de raffinement, (b) une base de données temporelle $TNET$ qui modélise l'état du malade et la thérapie en cours, (c) deux processus (*Interviewer* et *Reasoner*) qui gèrent les entrées et l'interrogation du système.

Le cycle d'inférence est mis en œuvre par le processus *Reasoner* qui fait intervenir un planificateur, le gestionnaire de la base et un module d'entrées/sorties. Le planificateur gère le raffinement hiérarchique de plans à partir des données issues de la base. Il invoque le module d'inférence pour déterminer quel plan exécuter en fonction des délais imposés par la tâche (thérapie) en cours. Les événements qui surviennent (état du malade) sont pris en compte par le gestionnaire de la base qui, par le module d'inférence, adapte la résolution en modifiant les contraintes de temps ou la tâche prioritaire.

I. Fergusen - TouringMachines est une architecture d'agents mobiles, autonomes et rationnels [Ferg92]. Elle est organisée en 3 couches (Réactivité - Planification - Modélisation) auxquelles sont associés un module de perception, un module d'action et des règles de contrôle. L'objectif visé par cette approche est de définir un contrôle rationnel qui adapte le temps de délibération à la criticité des événements traités.

La couche réactive est un module de déclenchement d'actions qui opère à partir d'un ensemble de règles "situations-actions" dépendantes du domaine. Outre le déclenchement des actions pour répondre aux événements critiques, cette couche désigne le point sur lequel doit porter l'activité de planification (*Focus of Attention*).

La couche de planification utilise une variante de l'approche du système Phoenix, chaque étape du plan est construite, puis exécutée avant de poursuivre la gestion du reste du plan.

I.2.7 Contrôle opportuniste

Cette approche du contrôle réactif se caractérise par la gestion des possibilités d'activation des différentes actions pilotées par le système. Elle a été mise en avant dans les systèmes blackboard.

◇ **Définition** *Possibilité d'activation*: Une possibilité d'activation est un ensemble de données de la résolution qui permet d'activer une action [Haye85].

Pour une action, plusieurs possibilités d'activation peuvent exister simultanément, elles correspondent à des données différentes. Inversement, un ensemble de données peut rendre activables plusieurs actions.

Objectifs de la gestion des possibilités d'activation

L'objectif recherché par la gestion des possibilités d'activation des actions est d'une part d'exploiter ces possibilités au niveau du contrôle et d'autre part de les utiliser comme représentation explicite d'une partie de la résolution.

Exploitation des possibilités d'activation L'exploitation des possibilités d'activation constitue l'une des mises en œuvre de la réactivité : elle reprend la phase d'évaluation des systèmes de production pour détecter pendant la résolution "ce qu'il est possible de faire". Les possibilités d'activation des actions sont enregistrées dans une structure d'agenda. Le rôle du contrôleur est alors d'analyser le contenu de cet agenda pour confronter "ce qu'il est possible de faire" avec "ce qu'il faut faire", c'est-à-dire avec les objectifs du système. Il doit donc sélectionner la possibilité d'activation la plus intéressante pour l'état courant de la résolution. Cette organisation du contrôle à partir d'un agenda est à la base des architectures de type blackboard. Un *KSAR* est une possibilité d'activation et le contrôle examine les *KSAR* pour mettre en œuvre le contrôle opportuniste. Cette exploitation des possibilités d'activation est à l'origine de la notoriété de cette approche. Elle semblait prometteuse dans le sens où elle permet un contrôle sophistiqué pour aborder les problèmes sans solution connue. La détection et l'exploitation des possibilités ont donc été vues comme la réponse à l'exploration fastidieuse de l'espace des solutions utilisé par certaines approches de la planification.

Evaluation et explication La gestion des possibilités d'activation des actions est intéressante car elle constitue une représentation explicite d'une partie de la résolution : elle permet d'accéder aux actions qui sont activables à partir d'une donnée d'une part, et aux données qui activent une action d'autre part. Ce lien entre les données et les actions permet d'analyser la résolution. Cette analyse intervient dans l'explication et dans l'évaluation de la résolution. Le travail de B. Lemaire [Lema92] concerne l'utilisation d'un système blackboard pour produire l'explication d'une résolution qui a été menée dans un système de production de règles, et non dans un système blackboard. Il n'utilise donc pas cet aspect du contrôle opportuniste qui, à partir de l'examen d'une possibilité non exploitée, permet de répondre à des questions de type *Pourquoi pas ?* L'évaluation de la résolution consiste à mesurer le rôle des actions et des données qui ont été utilisés.

Caractéristiques des applications visées

Ce mode de contrôle est intéressant pour les applications peu structurées, où les actions sont indépendantes les unes par rapport aux autres et où il existe peu d'enchaînements prédéfinis pour le déclenchement des actions. Les conflits d'actions sont alors nombreux et l'examen des possibilités permet de mettre en œuvre une résolution sophistiquée de ces conflits.

Les applications de robotique présentent des caractéristiques qui concernent le contrôle opportuniste. Entre toutes les actions que le robot est capable de déclencher, et entre toutes les missions qui lui sont confiées, il faut arbitrer pour privilégier les actions qui réalisent les missions prioritaires [Croc93].

La production de l'explication constitue également un domaine d'application favorable à l'utilisation du contrôle opportuniste. B. Lemaire stocke dans un agenda les différentes possibilités de déclencher les modules d'explication sur les différents éléments à expliquer. Les conflits sont nombreux et les stratégies d'explication varient en fonction des buts et des besoins de l'utilisateur. De même, en traitement du langage naturel, la gestion des ambiguïtés génère de nombreux conflits entre les modules disponibles. B. Bachimont a poussé assez loin l'utilisation du contrôle opportuniste pour la traduction de textes médicaux. Dans son système blackboard (ABACAB [Bach91]), les objets du blackboard sont les solutions possibles pour l'interprétation d'un élément du texte. Parce que plusieurs solutions sont possibles pour un même élément du texte, il a défini un système de maintien du raisonnement qui assure la cohérence et la convergence de la résolution.

Le modèle blackboard

La gestion (détection, stockage et analyse) des possibilités d'activation concernant un ensemble fini d'actions passe par l'organisation de ces actions autour d'une base de données commune qui contient les données de la résolution. Le modèle blackboard pour les architectures de contrôle est donc constitué de trois éléments : (a) un ensemble d'actions, (b) une base de données (centralisée ou distribuée) et (c) un module de contrôle pour arbitrer les conflits entre possibilités d'activation. Cette organisation des actions, indépendamment les unes des autres, autour de la base de données, a mis en évidence la possibilité de déclencher en parallèle les différentes actions. L'association du modèle blackboard et du parallélisme a donc contribué à la renommée du contrôle opportuniste.

Si la base de données est traditionnellement organisée en niveaux, le contrôle a été mis en œuvre de façons variées : le contrôle procédural applique un algorithme pour sélectionner la possibilité d'activation, le contrôle hiérarchique organise le contrôle en niveaux et le contrôle par blackboard confie cette tâche à un autre système blackboard (dit de contrôle) qui supervise le système blackboard de résolution. Ces différents travaux ainsi que les tendances de leurs évolutions sont présentés dans [Carv92].

Critique de cette approche

L'approche du contrôle opportuniste a été beaucoup critiquée, car elle est coûteuse en temps et en place mémoire. Dans les phases de la résolution où des enchaînements prédéfinis d'actions sont applicables, l'examen de l'agenda est pénalisant. Plusieurs variantes du contrôle opportuniste ont donc été proposées pour palier ce manque d'efficacité. Cependant l'image des systèmes blackboard reste attachée à celle d'outils pour la mise au point des stratégies de résolution. Ils sont peu envisagés pour une résolution en contexte opérationnel.

Plus encore, l'utilité du contrôle opportuniste dans une application réelle n'a pas été démontrée. Cependant, la gestion des possibilités d'activation reste intéressante pour l'évaluation et l'explication de l'activité interne du système.

I.3 Mécanismes d'inférences et temps réel

Un système à base de connaissances réactif doit apporter des réponses aux événements extérieurs. La qualité de cette réponse décroît avec le temps qui s'écoule. Il faut donc maîtriser le temps qui s'écoule entre l'occurrence de l'événement et sa prise en compte d'une part (voir section I.4), et le temps consacré à l'élaboration de la réponse d'autre part. Ce dernier temps est appelé temps de *délibération*. Dans un système à base de connaissances, la délibération est mise en œuvre par des mécanismes d'inférence dont le temps d'exécution doit donc être maîtrisé pour apporter une réponse à l'événement considéré dans un délai utile.

La maîtrise du temps d'exécution des mécanismes d'inférence est un problème spécifique des systèmes à base de connaissances [Hami92]. Il est dû à la nature de ces mécanismes d'inférence qui bien souvent effectuent une recherche dans des graphes complexes [Clan92]. Nous présentons dans cette section les différentes approches qui ont été proposées pour aborder ce problème.

I.3.1 Optimisation

L'optimisation des mécanismes d'inférence est motivée par le souci d'efficacité et par l'idée que l'amélioration constante des performances des processeurs permet d'envisager l'obtention de temps d'exécution compatibles avec la criticité des applications visées, et donc de pouvoir intégrer les mécanismes d'inférence dans les systèmes temps réel. Cet espoir est mis en défaut par les arguments suivants :

1. La complexité des applications (et donc celle des algorithmes) croît à mesure que la performance des processeurs augmente. Le “confort” d'exécution des inférences ne sera donc pas atteint.

2. Cette approche *temps borné* correspond à une vision réduite du temps réel. Un système temps réel ne doit pas seulement garantir une réponse aux événements en temps borné, mais il doit admettre la variable *temps* comme paramètre de contrôle : le délai spécifié pour un traitement varie dynamiquement, problème qui dépasse le cadre de l'optimisation [Bara92].

Les heuristiques appliquent des critères empiriques ou démontrés pour contrôler la recherche dans le graphe. Elles permettent d'obtenir une solution qui n'est pas nécessairement optimale.

Les techniques de compilation sont abondamment utilisées dans les systèmes de production. Un système de production associe une base de faits et des règles qui mettent à jour cette base. Ces techniques de compilation exploitent l'idée qu'une mise à jour de la base n'affecte qu'une partie bien identifiée de son contenu. Elles consistent donc à évaluer incrémentalement l'état de la base à partir des mises à jour et non à le réévaluer entièrement à chaque fois. Elles sont mises en œuvre au moyen d'un réseau qui associe les parties actions des règles aux structures de données qu'elles concernent. A partir de ce réseau, le déclenchement d'une règle affecte directement (sans recherche) les éléments de la base qui sont concernés par cette règle. Des algorithmes célèbres ont été proposés (TREAT, RETE) et ont été abondamment étudiés, adaptés et optimisés par rapport aux systèmes considérés. Une étude des techniques de compilation apparaît dans [Goel91], [Ghal88] et [Piqu92].

I.3.2 Traitement approximatif

Les techniques de traitement approximatif supposent que pour l'algorithme d'inférence considéré, il existe un ensemble (fini ou non) de solutions dont la qualité varie en précision, en certitude et en complétude. Elles consistent à rechercher un compromis entre le temps consommé par l'inférence et la qualité de la réponse obtenue [Dean91].

Dans un système à base de connaissances, ces techniques peuvent concerner trois classes d'approximation [Schw92]:

1. Approximation de la recherche : en limitant le parcours du graphe, le temps consommé par la recherche peut être réduit.

2. Approximation des données : lorsqu'elles sont trop nombreuses pour être toutes traitées, les données peuvent être regroupées en une unité (*agrégation en clusters*) qui est traitée à leur place. Cette approximation est principalement utilisable dans les applications où les données sont fortement bruitées.

3. Approximation des connaissances : il s'agit de privilégier un algorithme de traitement moins précis mais plus rapide que les autres algorithmes disponibles dans le système.

Les algorithmes *anytime* [Etzi89] sont des cas particuliers de traitement approximatif : ils peuvent être interrompus à tout moment pour fournir une réponse dont la qualité s'améliore avec le temps consacré au calcul. Les traitements approximatifs dans le cas général ne fournissent une réponse qu'à partir d'un seuil, et la qualité de cette réponse s'améliore par étapes successives. L'équipe de V. Lesser [Deck90] expérimente les traitements approximatifs dans une architecture blackboard pour le traitement du signal accoustique de véhicules mobiles.

La limitation de l'approche des traitements approximatifs apparaît lorsque plusieurs buts sont à prendre en compte simultanément. Un traitement approximatif ne permet pas de résoudre les conflits de buts ou les conflits de méthodes par rapport à plusieurs buts.

I.3.3 Compromis entre complexité et modélisation

La complexité des algorithmes utilisés dans un mécanisme d'inférence est due à la complexité du graphe parcouru et donc à la capacité de modélisation du système qui met en œuvre ce mécanisme. Il est possible d'obtenir une complexité polynomiale pour ces algorithmes en restreignant la capacité de modélisation. Cette solution au problème de la maîtrise des mécanismes d'inférence est défendue par R. Brachman et H. Levesque [Leve87] dans le cadre des systèmes de représentation des connaissances.

A partir de cette idée, de nombreux travaux ont été menés dans le domaine des langages terminologiques de type KL-ONE pour étudier la complexité introduite par chaque nouveau constructeur de termes [Nebe90]. La composante terminologique d'une logique descriptive est composée de termes logiques définis par des constructeurs de termes. La relation de *subsumption* est une relation de préordre définie sur les termes à partir de leurs structures. Le principal mécanisme d'inférence de ces systèmes de représentation des connaissances est la classification de concepts qui consiste à calculer pour un nouveau concept sa position dans la hiérarchie de subsumption des termes. L'introduction d'un nouveau constructeur de termes peut rendre ce calcul non polynomial, et donc incompatible avec les contraintes de temps exigées dans un système réactif. Cette complexité du mécanisme d'inférence est due à la capacité de modélisation obtenue en introduisant ce nouveau constructeur de termes. Elle rend le système de représentation inacceptable pour les applications réelles. La solution proposée pour retrouver une complexité polynomiale est d'interdire ce type de constructeurs. Elle se présente donc comme un compromis entre la capacité de modélisation du système de représentation des connaissances et la complexité de ses mécanismes d'inférence.

I.3.4 Rationalité limitée (rationalité économique)

J. Doyle s'oppose à l'approche de R. Brachmann. Il considère que la caractérisation des algorithmes d'inférence mis en œuvre dans les systèmes de représentation des connaissances ne constitue pas la solution au problème des temps de réponse du langage. Plus encore, il montre [Doyl91] que pour garantir une complexité polynomiale dans le pire des cas, il faut tellement restreindre le pouvoir de modélisation, que les systèmes de représentation des connaissances ainsi obtenus ne peuvent plus aborder les applications réelles. J. Doyle affirme donc que les notions de correction et de complétude logiques ainsi que la complexité dans le pire des cas ne permettent pas d'aborder les problèmes de la réactivité. Il leur oppose les notions de coût, de gain et d'utilité qui sont mises en avant dans la théorie de la décision.

Définition de la rationalité limitée

Le principe de rationalité a été introduit par A. Newell (voir III.1.1) pour régir le comportement d'un agent rationnel. Il concerne la rationalité logique par opposition à la rationalité limitée ou rationalité économique [Smit93]. Selon la rationalité logique tout ce qui peut être inféré doit l'être. Par opposition, la rationalité limitée n'infère que ce qui peut l'être à partir des ressources disponibles, en particulier en fonction du temps alloué.

◇ **Définition** *Rationalité limitée*: Un système à base de connaissances est rationnel s'il sélectionne les actions à exécuter de façon optimale par rapport à ses ressources disponibles [Doyl88].

La modélisation des connaissances dans un système à base de connaissances est incomplète et inconsistante par rapport à la réalité. Il y a une inertie entre sa mise à jour et les évolutions de l'environnement extérieur. De plus les ressources du système sont limitées, le temps est contraint pour prendre des décisions et la place mémoire manque pour stocker les informations. L'objectif de la rationalité limitée est de définir un comportement globalement optimal par rapport à ces ressources. Pour cela, il est nécessaire de déclencher des inférences pour gérer ces limites et pour contrôler les prises de décisions du système. Les travaux menés en rationalité limitée concernent donc le contrôle des mécanismes d'inférence dans un système à base de connaissances.

Il se pose immédiatement le problème de la récurrence infinie sur les niveaux de rationalité : il ne faut pas perdre trop de temps à "philosopher", il faut agir. En fait, ce problème ne se pose pas. Du point de vue théorique [Doyl89], la quantité d'information décroît rapidement avec le niveau de contrôle. Et concrètement, si les calculs qui interviennent dans le contrôle des mécanismes d'inférence sont coûteux en temps, ce coût n'est pas comparable avec celui des mécanismes du plus bas niveau. Le problème de la récurrence infinie n'est donc pas bloquant.

Rationalité limitée et théorie de la décision

La théorie de la rationalité limitée aborde le problème du choix entre plusieurs possibilités. Elle vise à effectuer de "bons choix", des choix qui sont optimaux par rapport aux ressources et elle s'attache à prouver cette optimalité. Pour cela, elle utilise les résultats de la théorie de la décision.

Plutôt que d'utiliser une préférence (un ordre partiel) sur l'ensemble des possibilités, la rationalité limitée est basée sur la notion d'utilité espérée. L'utilité est une fonction numérique qui s'applique sur les possibilités, telle que l'on ait $U(A) < U(B)$ si et seulement si la possibilité A est moins optimale que B et $U(A) = U(B)$ si et seulement si les possibilités A et B sont équivalentes. Le fait que l'utilité est une mesure additive permet d'introduire les probabilités associées aux possibilités et de passer de l'utilité ordinale (qui ordonne) à l'utilité espérée qui est cardinale : les quantités d'utilité espérée peuvent être ajoutées ou soustraites pour en exprimer d'autres.

I.4 Gestion des données issues de l'environnement

La gestion des données issues de l'environnement est un problème général des systèmes réactifs. Elle est rendue plus complexe dans les systèmes à base de connaissances à cause de l'absence de prédictibilité pour les algorithmes de traitement (voir section I.3).

I.4.1 Evénements et réponses

Entre le moment où un événement est détecté par le système à partir des données issues de l'environnement, et le moment où les effets de la réponse du système sont perceptibles, il s'écoule un intervalle de temps qui définit la fenêtre de réponse à l'événement. Cette fenêtre peut être contrainte par le temps nécessaire pour que la réponse soit effective. Il reste donc l'intervalle de temps qui permet de choisir cette action, le temps alloué pour la prise de décision.

Deux types d'événements sont observables : les événements qui correspondent à l'évolution lente d'une situation et ceux qui au contraire sont limités dans le temps et qui peuvent nécessiter une réponse rapide. C'est pour aborder ces deux types d'événements que le système *Phoenix* [Howe90] met en œuvre deux mécanismes d'inférence, l'un par réflexes pour les événements critiques et l'autre par planification pour les évolutions lentes.

I.4.2 Recueil des données

Le problème du recueil des données n'est pas particulier aux systèmes à base de connaissances réactifs, mis à part le fait qu'un traitement de haut niveau peut nécessiter la mise en œuvre d'un mécanisme sophistiqué pour le recueil des données. Ce mécanisme fait l'objet d'un intérêt tout particulier dans les travaux qui sont actuellement menés dans le projet *Guardian [Haye90]* de suivi médical en réanimation.

Les méthodes possibles pour l'échantillonnage des données issues de l'environnement sont les suivantes :

1. L'échantillonnage fixe consiste à recueillir les données à intervalles constants. Cette approche suppose d'une part que le système est capable de traiter tous les échantillons dans le délai constant, et d'autre part que les données ainsi échantillonnées sont significatives. Or une variation des données pendant la durée de traitement du dernier échantillon ne sera pas détectée.

2. L'échantillonnage moyenné permet de garantir des données significatives en fournissant comme valeur la moyenne des données observées pendant le traitement du dernier échantillon. Cependant, il subsiste un décalage entre les données réelles et la moyenne ainsi calculée.

3. L'échantillonnage par *polling* consiste à ne prélever l'échantillon que lorsque le traitement du précédent est terminé. Cette approche permet de garantir le traitement de tous les échantillons, mais la durée de traitement d'un échantillon n'étant pas prévisible dans le cas d'un mécanisme d'inférence, les événements qui surviennent pendant ce traitement ne seront pas pris en compte avant un délai inconnu. Cet aspect rend le comportement réactif du système difficile à prévoir.

4. Le seuillage fixe utilise des seuils fixes pour détecter les variations des données qui correspondent à des événements et pour déclencher des interruptions du système. Le seuillage fixe suppose que le comportement des données est bien connu car les variations en dessous des valeurs seuils peuvent correspondre à un disfonctionnement sans qu'il ne soit détecté. Dans le pire des cas, les oscillations des données autour des valeurs seuils vont provoquer une cascade d'événements détectés et donc d'interruptions du système.

Le projet *Guardian* [Ash93] s'efforce de concevoir une gestion performante des données en définissant un filtre dynamique [Wash89]. Les contraintes définies pour la gestion des données issues des capteurs dans un système réactif sont respectivement :

1. Le système doit adapter les traitements aux ressources disponibles et donc n'accepter que des volumes de données compatibles avec ses ressources.

2. Même s'il est en train de réaliser des inférences, le système doit être capable de prendre en compte les événements importants.

3. Les données doivent être surveillées de façon sélective : en détail pour les éléments en cours de traitement et suffisamment pour les autres afin de détecter les événements importants.

Ces contraintes expriment le fait que la gestion des données dans un système à base de connaissances réactif requiert un niveau de complexité minimum pour alimenter de façon performante les mécanismes d'inférence. L'approche proposée consiste à associer le seuillage et l'échantillonnage. Le seuillage utilise des seuils variables qui sont calculés à partir des valeurs précédentes. Les intervalles d'échantillonnage sont également variables.

I.5 Choix d'une architecture

A partir de l'étude des solutions possibles pour la conception d'un système à base de connaissances réactif qui pilote des modules algorithmiques, nous expliquons ici les choix que nous avons effectués pour notre système TRAM.

I.5.1 Historique de TRAM

Une première version de TRAM [Koen89] a été réalisée en 1989 pour contrôler un robot autonome mobile. Il s'agit d'un système blackboard de type BB1 [Haye85] développé en Lisp. Dans cette version, les actions sont des règles de production dont la condition est décomposée en plusieurs parties. La base de données est organisée en plusieurs niveaux d'abstraction et gère les possibilités d'activation des actions. L'arbitrage entre les différentes actions activables est fait par un module de contrôle qui est organisé sous la forme d'un deuxième système blackboard dit de contrôle.

La réécriture de TRAM en langage C a été motivée par les contraintes liées aux systèmes embarqués. A cette occasion, nous avons redéfini les objectifs du système. Il ne s'agit plus de concevoir le système de contrôle de navigation d'un robot mais de définir un $\mathcal{ERP}$ réactif en informatique scientifique et technique : contrôle non destructif, détection sous-marine et robotique.

L'organisation du système TRAM a été redéfinie, car plusieurs caractéristiques de la première version sont inappropriées pour un $\mathcal{ERP}$ réactif.

- La définition des actions sous la forme de règles de production est moins intéressante que l'utilisation des tâches et des méthodes pour modéliser le comportement de modules algorithmiques ou pour mettre en œuvre des plans d'actions.

- L'organisation de la base en une hiérarchie de niveaux n'autorise que deux dimensions pour la modélisation : la dimension d'abstraction entre les niveaux et le séquencement à l'intérieur de chaque niveau. B. Bachimont [Bach92] a montré que cette limitation ne convient pas à la résolution de problèmes et qu'elle rend artificielle la relation d'abstraction entre des niveaux qui modélisent des entités indépendantes. Il préconise une organisation de la base en plusieurs dimensions.

- Le contrôle de type BB1 est élégant, réputé souple et puissant, mais rend difficile la définition des principes de résolution. Les plans d'actions qui sont utilisés en informatique scientifique pour ordonner les modules et décrire le flot des données entre ces modules ne peuvent pas être directement pris en compte dans ce type de contrôle : l'expression du séquencement seul oblige à associer aux actions des coefficients numériques pour définir une priorité. Si le séquencement varie pendant la résolution, alors il faut relativiser les coefficients, ce qui rend complexe la gestion du contrôle.

- Le premier système TRAM ne prend pas en compte les contraintes liées aux systèmes¡réactifs telles que la maîtrise des ressources disponibles. Il ne définit pas un mécanisme rationnel de sélection des actions.

Ces différents points nous ont donc conduit à redéfinir l'organisation du système pour la nouvelle version de TRAM qui n'est plus un système blackboard de type BB1.

I.5.2 Contraintes

TRAM est un système à base de connaissances réactif dont le rôle est de piloter des modules algorithmiques en résolution de problèmes. Nous reprenons l'organisation en couches des systèmes à base de connaissances réactifs. Ces couches sont respectivement la réactivité, la planification et la modélisation de l'état courant de la résolution. Cette organisation est générique et peut être adaptée aux besoins d'un $\mathcal{ERP}$. Cette section présente les contraintes imposées à TRAM et montre comment elles nous ont conduits à choisir ce type d'organisation pour la conception de notre système.

Réutilisabilité

TRAM est destiné à des domaines d'application pour lesquels des solutions logicielles existent indépendamment des systèmes à base de connaissances. Par le terme de réutilisabilité nous désignons les contraintes qui imposent au système de s'adapter à ces solutions et non l'inverse. A. Newell [Newe93] établit le bilan des expériences menées dans le développement d'applications à partir d'un système à base de connaissances avec des modules logiciels existants. Il montre bien que le système à base de connaissances ne peut pas prétendre imposer ses contraintes à ces modules. Aussi, la contrainte de compatibilité avec l'existant est-elle très forte (à la mesure des millions de lignes de codes en Fortran contenues dans les modules algorithmiques qui sont visés par ce type de système).

La réutilisabilité a comme première conséquence l'intégration sans modification des modules algorithmiques dans l'environnement TRAM. Cette intégration est gérée par l'approche client/serveur de TRAM qui est présentée dans la section IV.2. Les éléments de résolution qui doivent être pris en compte sans modification ne concernent cependant pas que les modules algorithmiques. Les schémas de résolution doivent également être respectés. Ce sont tous les enchaînements d'actions qui sont éprouvés pour résoudre un problème donné dans des situations bien identifiées. Ces enchaînements se traduisent naturellement sous la forme de plans d'actions qui définissent la décomposition des tâches composites en sous-tâches. De même, pour décrire les connaissances relatives aux aspects réactifs de la résolution, il est naturel de faire référence à la situation courante et aux traitements des événements qui doivent être gérés à partir de cette situation. Cette approche de l'expression du contrôle réactif identifie bien le rôle d'un diagramme d'états dans la définition du contrôle tel qu'il a été décrit dans l'organisation de plusieurs systèmes à base de connaissances réactifs (voir I.2).

Explication et évaluation

TRAM doit fournir une *trace* qui permet d'évaluer et d'expliquer l'activité du système après la résolution. Le terme de *trace* est utilisé sans préjuger de sa signification ni de son éventuelle insuffisance vis-à-vis de l'explication.

Les domaines d'application visés peuvent être peu structurés, et bien que de nombreuses solutions partielles existent pour traiter une partie d'un problème ou un problème dans une situation restreinte, TRAM a pour objectif de constituer une plateforme d'évaluation des rôles respectifs des différents éléments de solution dans les différentes catégories de situations. Pour satisfaire ces objectifs, nous avons donc mis en œuvre la détection et la gestion des possi¡bilités d'activation au moyen des configurations d'activation (III.6). L'évaluation des possibilités d'activation permet en effet de mesurer le rôle d'une action donnée dans les différentes phases de la résolution pour profiter des possibilités non exploitées ou pour détecter et traiter les cas d'actions peu activables. De plus, vis-à-vis de l'explication, la gestion des possibilités d'activation permet de répondre à des questions du type "Pourquoi pas?" en analysant les données qui permettaient éventuellement d'activer une action.

L'intérêt des tâches pour l'explication fait partie des raisons qui ont favorisé leur introduction dans les systèmes à base de connaissances [Chan86]. L'approche du contrôle réactif par un automate d'états finis (voir I.2) permet une représentation explicite des connaissances liées à l'aspect réactif de la résolution, elle contribue donc à rendre le système explicable.

Efficacité

Sans que les contraintes de temps atteignent des criticités trop fortes dans un $\mathcal{ERP}$ réactif, il faut cependant que la résolution soit efficace, en particulier dans les phases où des solutions connues doivent être utilisées. Cette contrainte est en opposition avec la gestion des possibilités d'activation qui pénalise nécessairement le temps de réaction du système puisqu'il doit mener une activité de détection et de stockage des possibilités d'activation parallèlement à la résolution. Pour réduire cette dégradation des performances, nous avons proposé des algorithmes d'évaluation incrémentale de l'état courant de la base à partir de ses mises à jour par les modules algorithmiques (II.6).

En ce qui concerne la planification, nous privilégions l'approche de la rationalité limitée qui consiste à maîtriser le temps utilisé pour effectuer les calculs qui interviennent dans ce module. La planification étant mise en œuvre au moyen des tâches et des méthodes, nous proposons dans le chapitre III un algorithme de sélection rationnelle de méthodes pour réaliser la tâche courante.

Contrôle opportuniste

Nous avons abordé la problématique de la réactivité dans un $\mathcal{ERP}$ dans le but de définir un système autonome, et dans ce contexte, l'opportunisme apporte peu. Par contre, par rapport aux aspects coopératifs de l'utilisation d'un $\mathcal{ERP}$, le contrôle opportuniste peut être intéressant. En effet, si TRAM est couplé avec une bibliothèque de traitement d'images et qu'il est mis à la disposition d'un utilisateur spécialiste du domaine de la vision, alors la gestion des possibilités d'activation permet de proposer des méthodes qu'il ne connait pas, mais qui sont utilisables pour le problème qu'il traite. Plus encore, de l'aveu même des utilisateurs des environnements de traitement d'images, devant la pléthore de modules disponibles, ils ne connaissent qu'une partie de ces modules, qu'ils utilisent spontanément pour aborder les problèmes posés.

Dans le contexte de la coopération, le rôle de l'opportunisme est donc de casser les habitudes de l'utilisateur.

Ayant choisi de gérer les possibilités d'activation dans un système réactif, la mise en œuvre du contrôle opportuniste ne représente pas un surcoût prohibitif.

I.5.3 Organisation générale de TRAM

A partir des contraintes qui sont imposées, nous avons organisé le système TRAM en trois niveaux qui correspondent respectivement à la réactivité, à la gestion des tâches et à la modélisation de l'état courant de la résolution. La couche réactive est définie à partir d'un automate d'états finis. Ces états désignent chacun une tâche principale dont la couche de gestion des tâches assure l'exécution. La base contient la modélisation de l'état de la résolution, elle est dotée d'un mécanisme de détection et de gestion des possibilités d'activation. Ces possibilités d'activation peuvent définir des événements qui sont alors pris en compte par la couche réactive pour mettre en œuvre le mode de contrôle opportuniste. L'organisation générale de TRAM est décrite dans la figure I.3.

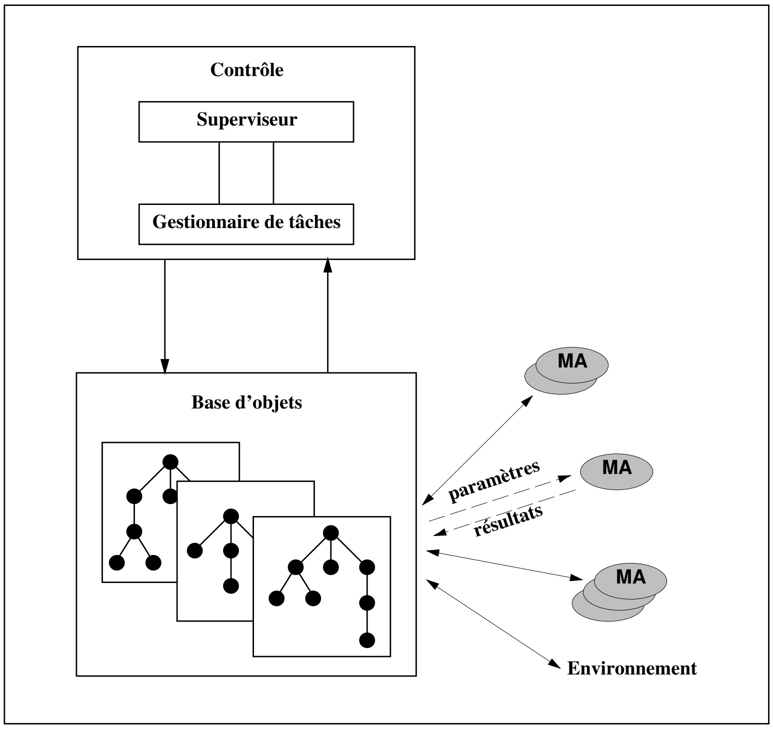

FIG. I.3 - *Organisation générale de TRAM.*

Le flot des données entre les différents composants est le suivant (Figure

I.4) :

La détection des événements (evt) est assurée par le gestionnaire de la base pour les événements qui concernent les données de la résolution et par le gestionnaire de tâches pour ceux qui concernent l'exécution des tâches.

Le comportement réactif par rapport à ces événements est mis en œuvre par le superviseur qui, pour répondre à ces événements, désigne une nouvelle tâche principale (t_p) à réaliser.

La formulation (f) des méthodes (plans d'action ou modules algorithmiques) associées à la tâche est faite par le gestionnaire de tâches.

La sélection d'une méthode (m) parmi les différentes possibilités est effectuée par le gestionnaire de la base. L'identificateur de cette méthode est retourné au gestionnaire de tâches.

Si la méthode sélectionnée est un plan d'actions, la gestion des sous-tâches est faite par le gestionnaire de tâches. S'il s'agit d'un module algorithmique, c'est le gestionnaire de la base qui contrôle son exécution en transmettant les paramètres (p) et en mettant à jour le contenu de la base à partir des résultats (r). Le compte rendu (cr) du gestionnaire de la base contient le rapport d'exécution de l'action et les événements qui résultent de l'évaluation de l'état courant de la base.

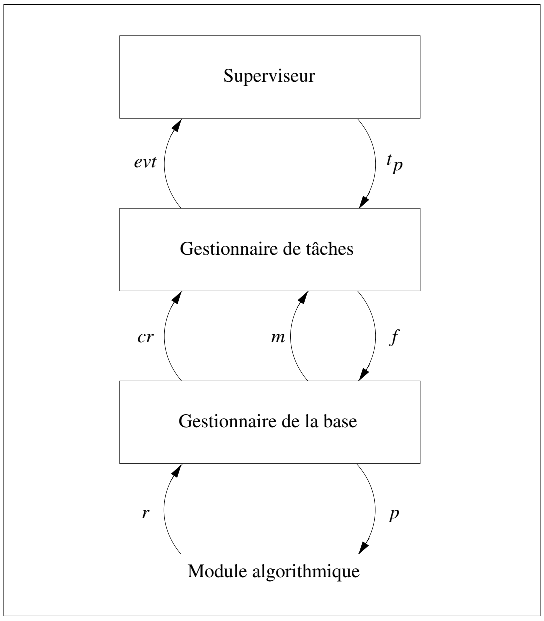

FIG. I.4 - *Flot des données entre les différents composants de TRAM.*

I.6 Conclusion

Nous avons présenté dans ce chapitre l'organisation générale d'un $\mathcal{ERP}$ réactif. Un superviseur défini à partir d'un automate d'états finis désigne la tâche qui est réalisée par le gestionnaire de tâches. L'organisation de la base de données qui contient la modélisation de l'état courant de la résolution est traitée dans le chapitre II. Parmi les solutions possibles pour maîtriser les inférences dans un système à base de connaissances réactif, nous privilégions la rationalité limitée en définissant dans le chapitre III un mécanisme de sélection rationnelle de méthodes.

Chapitre II

Base d'objets

Ce chapitre étudie le problème de la modélisation des entités de la résolution dans un $\mathcal{ERP}$ réactif [Chai92a]. Nous proposons un modèle objet pour organiser la base de données d'un $\mathcal{ERP}$ réactif. La caractérisation de l'état courant de la résolution est faite localement par le mécanisme de classification d'instances et globalement par l'évaluation des configurations. Cette caractérisation de la base permet de détecter les événements qui intéressent la couche réactive du contrôle.

Le rôle de la base d'objets dans un $\mathcal{ERP}$ réactif est de modéliser l'état de la résolution. Cette modélisation intéresse le contrôle et les actions qui sont activées à partir des données de la résolution. Elle nécessite deux fonctions :

- *Base de données* : la base stocke et restitue les données qui sont utilisées et produites par les actions. Cette fonction est celle d'une base de données.

- *Caractérisation de la résolution* : la base est un composant actif du système. Sa fonction est alors de caractériser l'état courant de la résolution. Cette caractérisation concerne d'une part les données et d'autre part des ensembles de données. Elle doit ainsi détecter des situations particulières qui intéressent le contrôle ou qui permettent d'activer des actions.

La modélisation de la résolution est faite dans un système de représentation par objets. La caractérisation de l'état courant de la résolution est réalisée à partir des mécanismes d'inférence qui sont mis en œuvre dans la base d'objets. L'architecture de contrôle devient ainsi un système de production d'objets dont les mises à jour sont effectuées par les actions qui sont déclenchées.

Ce chapitre présente tout d'abord les différentes approches envisageables pour modéliser la résolution dans un système de résolution de problèmes. Le modèle objet est ensuite décrit de manière informelle et formelle. Puis les actions de mise à jour de la base sont expliquées ainsi que leurs effets sur le comportement des objets. Enfin, le mécanisme d'évaluation de l'état de la base est défini et les algorithmes sont décrits.

II.1 Approches de la modélisation en résolution de problèmes

Plusieurs approches sont envisageables pour modéliser l'état de la résolution dans un $\mathcal{ERP}$. Nous présentons ci-dessous les grandes familles de systèmes de représentation des connaissances, en explicitant leurs caractéristiques par rapport au rôle de la base dans le système TRAM (voir chapitre IV).

II.1.1 Représentation des connaissances par objets

Les systèmes de représentation des connaissances par objets sont basés sur les notions de classes et d'instances [Rech85]. Leur spécificité est d'autoriser la migration d'une instance d'une classe vers une autre. Les classes sont organisées selon une relation d'ordre partiel, la spécialisation. La classification d'un objet consiste à déterminer toutes les classes dont il est instance. Ce mécanisme d'inférence permet de caractériser une instance en la rattachant aux classes les plus spécialisées. D'autres mécanismes d'inférence sont mis en œuvre dans ce type de représentation des connaissances, comme l'attachement procédural, l'inférence par valeur et l'inférence par défaut.

La puissance du mécanisme d'inférence par classification d'instances est intéressante pour caractériser l'état courant de la résolution. La classification peut être utilisée pour détecter les événements dans la base et pour assurer la coopération entre des actions indépendantes. La caractérisation d'une partie de la base peut être réalisée par les objets complexes et les objets composites [Mari90], mais ce mécanisme ne concerne que les éléments explicitement liés avec l'instance classée et ne permet pas de détecter des événements plus larges. Pour caractériser globalement la base, il faut donc concevoir un nouveau mécanisme.

II.1.2 Logiques descriptives - Langages terminologiques

Les langages terminologiques ou logiques descriptives sont issus du langage KL-ONE développé par R. Brachman [Brac85]. Ils privilégient la composante logique d'un système de représentation des connaissances en définissant les concepts sous la forme de formules logiques. Leur principal mécanisme d'inférence est le *classifier* qui, pour un concept donné, recherche tous les concepts qui sont en relation avec lui pour la relation de subsumption [Wood91]. Si des travaux très formels sur la complexité de la subsumption ont été menés (voir I.3.3), les langages terminologiques restent peu adaptés aux applications. Cet aspect constitue la principale direction des travaux en cours : l'absence de structure d'objets est actuellement compensée par l'introduction des domaines concrets dans le formalisme [Baad91] et les résultats de complexité sont contournés pour aborder les problèmes du temps réel [Nebe90].

Le système CLASP développé par J. Yen et R. McGregor [Yen91] est un système de production qui met en œuvre la subsumption. Il montre bien la puissance obtenue en combinant un mécanisme de caractérisation (la subsumption) et les aspects des systèmes de production. Nous reprendrons cette idée pour définir un système de production d'objets.

II.1.3 Bases de données

Les évolutions actuelles [Hurs93] [Kim90] [Freu90] des bases de données les rapprochent des bases de connaissances. De nombreux travaux en bases de données concernent l'intégration de mécanismes d'inférence. Cette tendance est motivée par le besoin croissant de mettre en œuvre des traitements de haut niveau pour les données qui sont gérées.

Les inférences sont principalement définies par des règles [Hans92]. Si les étiquettes sont nombreuses, deux familles de systèmes se distinguent selon que le contrôle des règles est déterministe ou non [Abit91].

(a) Bases de données déductives

Les bases de données déductives associent un contrôle déterministe aux règles qui sont activables dans la base de données. Les règles activables sont déclenchées par un programme extérieur qui les utilise pour déduire de nouvelles données et donc mettre à jour le contenu de la base.

Extension du modèle relationnel : Les bases de données déductives mettent en œuvre des mécanismes d'inférence à partir de bases de données relationnelle. Pour cela, elles combinent la programmation logique avec le modèle relationnel. L'approche utilisée consiste à interpréter une base de données relationnelles comme une théorie du premier ordre. Le nom d'une relation définit un prédicat n-aire et les n-uplets de cette relation sont des formules atomiques qui instancient ce prédicat. D'autres formules peuvent être construites et l'introduction de règles permet de définir des relations dérivées. L'hypothèse du monde clos (tout fait non démontrable dans la base est considéré comme faux) permet de déduire de nouveaux théorèmes et donc d'inférer de nouvelles données.

Les langages de règles sont généralement associés à Prolog. Le plus connu est le langage *Datalog* [Ceri89] qui définit une restriction de Prolog. Son utilisation dans une base de données relationnelle permet d'exprimer les requêtes, d'utiliser la base de données pour gérer la persistance des objets.

Bases de données déductives orientées objets : Les règles sont associées aux objets dans les nombreuses bases de données orientées objets qui sont proposées. Si des *règles d'or* [Hurs93] permettent de caractériser l'aspect orienté objet d'une base, les mécanismes mis en œuvre sont variés [Bert92].

(b) Bases de données actives

Dans les bases de données actives, le contrôle des règles est non déterministe : toutes les règles activables sont déclenchées, dotant ainsi la base d'un mécanisme de détection et de réaction à des événements [Gatz93] qui est celui des systèmes de productions à base de règles [Forg81]. Cette réaction est automatique, elle ne nécessite pas d'intervention extérieure.

Les bases de données actives sont utilisées pour réagir à des situations particulières qui dépendent de l'application : pour réagir à un instant donné, pour gérer des contraintes d'intégrité, pour contrôler l'accès aux données ou pour propager automatiquement les mises à jour de la base. Les données créées par les règles de production sont des données dérivées.

Tout d'abord définies comme des extensions du modèle relationnel, les bases de données actives sont actuellement développées à partir de modèles objets [Kim90]. Le modèle objet F2 [Falq90] [Esti93] gère des "contextes" qui constituent des filtres analogues à ceux que nous souhaitons définir pour gérer les possibilités d'activation dans la base.

II.1.4 Choix d'une représentation par objets

Pour modéliser l'état de la résolution dans notre système, nous avons défini un modèle objet qui combine les mécanismes d'inférence de deux types de représentation par objets :

- Le mécanisme de classification d'instances propre aux systèmes de représentation par objets permet de caractériser les objets de la base.

- La détection d'événements dans les bases de données actives permet de caractériser globalement l'état de la base.

II.2 Description du modèle objet proposé

Notre modèle objet reprend les éléments des représentations par objets (SHIRKA [Rech85], TROPES [Mari90]). Les *objets* semblables sont regroupés dans des concepts. Un *concept* définit la structure de ses objets par des *attributs*. Les attributs sont de deux types : le contenu d'un attribut structurel est une valeur, celui d'un attribut relation est une liste de références à d'autres objets de la base. Un concept définit donc un ensemble d'objets de même structure. Un objet de ce concept est caractérisé à partir des valeurs de ses attributs. Ces valeurs peuvent être en partie inconnues. Une *classe* est un sous-ensemble d'objets du concept dont les caractéristiques sont semblables. Cette similitude s'exprime par des *conditions d'appartenance* qui portent sur les valeurs des attributs des objets de la classe. Une classe est donc spécifiée par sa condition d'appartenance. Un objet est instance d'une classe si les valeurs de ses attributs satisfont la condition d'appartenance de la classe. L'extension d'une classe est l'ensemble de toutes ses instances.

Les conditions d'appartenance des différentes classes d'un concept peuvent être comparées, et une relation d'ordre partiel est définie à partir de cette comparaison. Cette relation d'ordre partiel organise l'ensemble des classes d'un concept sous la forme d'une *taxonomie* de classes ou taxonomie de *spécialisation*. Une classe est une sous-classe d'une classe c si elle fait partie du même concept et si sa condition d'appartenance est une restriction de celle de c. La classe *racine* de la taxonomie contient tous les objets du concept, les classes *terminales* contiennent les objets dont les propriétés sont les plus contraintes. La relation de spécialisation implique l'inclusion des extensions des classes.

Les caractéristiques souhaitées pour le modèle objet sont les suivantes :

Modèle statique : Le schéma de la base n'évolue pas pendant la résolution. Les définitions des concepts et des classes sont faites lors de la phase d'initialisation et restent valides pour la phase opérationnelle. L'aspect dynamique de la base est donc limité aux évolutions des objets qu'elle contient.

Absence d'allocation dynamique : Pour éviter les échecs pendant la résolution, le modèle objet doit prendre en compte les limitations de la ressource place mémoire. Les allocations dynamiques sont donc réalisées pendant la phase d'initialisation. Un concept est ainsi défini comme un réservoir d'objets qui sont alloués ou libérés. Lorsque tous les objets d'un concept sont alloués, des techniques d'oubli [Stre91] sont mises en œuvre pour libérer certains des objets de ce concept. La gestion de l'allocation et de la libération des objets est explicitée dans le modèle formel. Cette caractéristique est due au fait que le système est embarqué, ce qui exclut toute erreur système qui pourrait intervenir après la phase d'initialisation.

Spécialisation et absence d'héritage : Le modèle objet proposé met en œuvre la spécialisation de classes, mais pas l'héritage entre les classes. Les deux notions sont fortement liées : une sous-classe hérite les propriétés de sa sur-classe.

Cependant, en définissant les attributs au niveau du concept, l'héritage d'attributs entre les classes n'intervient plus. Seul reste l'héritage des facettes qui contraignent ces attributs pour chaque classe. La condition d'appartenance d'une classe est donc construite à partir de celle de sa sur-classe directe. Le modèle objet étant statique, nous considérons le résultat de l'héritage. La condition d'appartenance est entièrement définie au niveau de chaque classe et elle est une restriction de celle de sa sur-classe directe.

Absence de multi-spécialisation : Pour des raisons de simplicité, le modèle objet proposé ne prend pas en compte la multi-spécialisation : une classe ne peut avoir qu'une seule sur-classe directe. Cette limitation est très restrictive, mais elle est acceptable car les algorithmes proposés pour la caractérisation de la base peuvent être facilement étendus pour prendre en compte les particularités de la multi-spécialisation. De plus, des travaux sur la représentation multi points de vue sont actuellement en cours [Mari90]. Selon cette approche, un concept est décomposé en plusieurs points de vue, chacun organisant une partie des classes du concept sous la forme d'une taxonomie sans multi-spécialisation. Notre travail se place dans la perspective de cette approche.

Méthodes : Il est possible d'associer aux objets des méthodes (au sens des langages orientés objets). Mais notre approche consiste à sortir les méthodes de la base pour les soumettre au module de contrôle de l'architecture. Seules les conditions d'appartenance attachées aux classes correspondent à la notion de méthode des modèles orientés objets. Mais il n'est pas utile dans ce travail d'expliciter cet aspect.

La section suivante (II.3) présente une description formelle du modèle objet. Puis les actions de mise à jour de la base sont définies (II.4) et le mécanisme de caractérisation globale de la base par les configurations est décrit (II.5) et les algorithmes de calcul incrémental sont donnés (II.6).

II.3 Définition formelle du modèle objet

Cette section propose une formalisation du modèle objet. La partie structurelle des objets est formalisée à partir des travaux de L. Cardelli [Card85] [Card91] et A. Borgida [Borg89a]. Ce modèle formel de notre base d'objets sera ensuite utilisé pour exprimer et étudier nos algorithmes d'évaluation et de mise à jour de la base.

II.3.1 Objets et valeurs

Le modèle objet est basé sur les ensembles suivants :

- un ensemble fini de *domaines concrets* $D_1, ..., D_n et \mathcal{B} = \{vrai, faux\}$ avec $\mathcal{D} = \cup_{i=1}^{n} D_i \cup \mathcal{B}$.
- un ensemble fini A de noms d'*attributs*.
- un ensemble fini ID d'*identificateurs*, avec **Err** $\in ID$.
- un ensemble fini N_{Cl} de *noms de classes*.
- un ensemble fini N_{Cp} de *noms de concepts*.

Notre description formelle est basée sur la manipulation des identificateurs d'objets, de classes et de concepts. Les actions extérieures qui mettent à jour la base ne peuvent en effet accéder qu'aux identificateurs et non aux objets eux-mêmes.
◇ **Définition** *Valeurs*: L'ensemble $\mathcal{V}$ regroupe les valeurs définies par les règles de construction suivantes [1]:

$$\frac{v \in D_i \quad a \in A}{E \vdash < a : v > \ valeur} \qquad \frac{I \subset ID \quad a \in A}{E \vdash < a : I > \ valeur}$$

$$\frac{E \vdash v \ valeur \quad a \in A \quad v' \in D_i}{E \vdash < v, a : v' > \ valeur}$$

$$\frac{E \vdash v \ valeur \quad a \in A \quad E \vdash I \subset ID}{E \vdash < v, a : I > \ valeur}$$

Les valeurs sont *élémentaires*, *ensemble d'objets* ou *structurelles*. L'environnement statique E est défini plus loin.
◇ **Définition** *Objet*: Un objet est un quadruplet $o = (id, n_{cp}, n_{cl}, v)$ où $id \in ID$ est son identificateur, $n_{cp} \in N_{Cp}$ est le nom de son concept, $n_{cl} \in N_{Cl}$ est le nom de la classe à laquelle il est rattaché, et $v \in \mathcal{V}$ est sa valeur. Ils seront respectivement dénotés par $o.id$, $o.cp$, $o.cl$ et $o.val$. L'ensemble des objets de la base est noté $\mathcal{O}$.

[1] La règle d'inférence $\frac{E_1 \vdash e_i}{E_2 \vdash e_c}$ se lit : “si les expressions e_i sont vraies dans l'environnement E_1, alors l'expression e_c est vraie dans l'environnement E_2”.

II.3.2 Concepts et structure

La notion de concept constitue la partie structurelle du modèle objet. Un concept définit la structure commune à plusieurs objets.

$\diamond$ **Définition** *Concept*: Un concept est un couple $cp = (n_{cp}, \mathcal{S})$ où $n_{cp} \in N_{Cp}$ est le nom du concept, $\mathcal{S}$ est la structure de ses objets. Ils sont notés respectivement $cp.n$ et $cp.struct$. L'ensemble des concepts de la base est noté $\mathcal{CP}$.

$\diamond$ **Définition** *Structure*: Une structure est définie par les règles de construction suivantes :

$$\frac{a \in A}{E \vdash <>.a : \perp} \qquad \frac{E \vdash s\ structure \quad a \in A \quad E \vdash s.a : \perp}{E \vdash < s, a : D_i >.a : D_i}$$

$$\frac{E \vdash s\ structure \quad a \in A \quad E \vdash s.a : \perp \quad n_{cp} \in N_{Cp}}{E \vdash < s, a : n_{cp} >.a : n_{cp}}$$

L'expression $E \vdash s.a : \tau$ se lit "dans l'environnement E, s est une structure bien formée dans laquelle l'attribut a est de type τ".

$\diamond$ **Définition** *Type d'un attribut*: Le type d'un attribut dans une structure est le domaine concret ou le nom du concept qui est associé à cet attribut dans la structure. Si l'attribut a n'est pas défini dans la structure s alors son type est $\perp$. L'attribut a est de type τ dans la structure s se note : $s.a : \tau$.

Remarque : Le modèle de données ainsi défini n'attache pas de sémantique particulière pour les attributs de type relation qui font qu'un objet peut contenir comme valeur d'attribut une ou plusieurs références à d'autres objets. Cette absence de sémantique interviendra pour la définition de la classification d'instances (II.3.4).

Les règles construction des structures seraient plus complexes si la position des attributs dans la structure avait une importance (voir [Card91]).

Vérification de types

Les valeurs utilisées dans les objets doivent être du type qui est associé au concept de cet objet. Les règles de vérification des types pour les valeurs sont les suivantes :

$$\frac{\vdash E\ env}{E \vdash \emptyset :<>} \qquad \frac{E \vdash v : s \quad a \in A \quad v' \in D_i}{E \vdash < v, a : v' >:< s, a : D_i >}$$

$$\frac{E \vdash v : s \quad n_{cp} \in N_{Cp} \quad a \in A \quad \vdash E(id_i) = o_i \quad E \vdash o_i.cp = n_{cp}}{E \vdash < v, a : \{id_1, \ldots, id_n\} >:< s, a : n_{cp} >}$$

◇ **Définition** *Objet consistant avec un concept*: Un objet est consistant avec un concept si sa valeur est du type de la structure de ce concept.

$$\frac{E \vdash o.cp = n_{cp} \quad \vdash E(n_{cp}) = cp \quad E \vdash cp.struct = s \quad E \vdash o.val : s}{E \vdash o \approx cp}$$

Par extension, on utilisera également la notation $id \approx cp$ pour indiquer que l'objet d'identificateur id est consistant avec le concept cp.

Exemple de concept

En contrôle non destructif, le concept des défauts est le suivant :

```
CP-defaut { nature : nat-defaut ;
            hauteur : reel ;
            largeur : reel ; }
```

où le type *nat-defaut* est un type énuméré (*trou, fissure, protubérance*) et où les attributs *hauteur* et *largeur* sont interprétés en fonction de la nature du défaut comme sa profondeur ou sa hauteur et comme sa largeur ou son diamètre.

II.3.3 Classes et conditions d'appartenance

L'aspect dynamique du modèle objet a pour objectif de caractériser les états de la résolution. Cette caractérisation est faite en regroupant les objets qui partagent les mêmes propriétés dans des classes.

Classes

Les aspects ensemblistes du modèle objet sont mis en œuvre par les classes qui définissent les sous-ensembles d'objets qui interviennent dans la résolution. Les propriétés communes aux objets d'une classe définissent la condition d'appartenance de cette classe.
◇ **Définition** *Classe*: Une classe c est un quadruplet $c = (n_{cl}, p, n_{cp}, \theta)$ où $n_{cl} \in N_{Cl}$ est le nom de la classe, $p \in N_{Cl}$ est le nom de sa surclasse, $n_{cp} \in N_{Cp}$ est le nom de son concept, et θ est la condition d'appartenance d'un objet à la classe. Ils sont respectivement dénotés par $c.n$, $c.p$, $c.cp$ et $c.\theta$. L'ensemble des classes de la base est noté $\mathcal{CL}$.

Condition d'appartenance d'une classe

La condition d'appartenance d'une classe spécifie les propriétés des objets qu'elle contient.

Syntaxe des conditions : La syntaxe du langage de définition des conditions d'appartenance aux classes est donnée par la grammaire suivante :

$$
\begin{array}{lcl}
<\!\mathit{condition}\!> & ::= & <\!\mathit{prédicat}\!>^{*}\,; \\
<\!\mathit{prédicat}\!> & ::= & <\!\mathit{objval}\!>\ <\!\mathit{op}\!>\ <\!\mathit{valeur}\!> \\
 & | & <\!\mathit{objval}\!>\ <\!\mathit{op}\!>\ <\!\mathit{objval}\!> \\
 & | & <\!\mathit{relation}\!> \in\ <\!\mathit{classe}\!>\,; \\
<\!\mathit{objval}\!> & ::= & <\!\mathit{attribut}\!> \\
 & | & <\!\mathit{relation}\!>.\mathrm{val}.<\!\mathit{attribut}\!>\,;
\end{array}
$$

Les opérateurs $< op >$ sont du type $op : \mathcal{V} \times \mathcal{V} \longrightarrow \mathcal{B}$. Cette syntaxe permet d'accéder aux attributs des objets en relation avec l'instance considérée dans le test.

Exemple La condition d'appartenance de la classe des fissures larges est la suivante :

COND-fissure-large = (nature == fissure)(largeur $\geq s_{fl}$)

où la constante numérique s_{fl} spécifie le seuil en largeur des fissures larges.

Calcul des conditions : Le calcul des conditions est défini par les prédicats et les formules :

si $r \in A$, si $n_{cl} \in N_{Cl}$, alors $(r \in n_{cl})$ est un prédicat.

si $r \in A$, si $a \in A$, si $v \in \mathcal{V}$ et si $op : \mathcal{V} \times \mathcal{V} \longrightarrow \mathcal{B}$, alors $(r.val.aopv)$ est un prédicat.

si $r_1, r_2 \in A$, si $a_1, a_2 \in A$ et si $op : \mathcal{V} \times \mathcal{V} \longrightarrow \mathcal{B}$ alors $(r_1.val.a_1\, op\, r_2.val.a_2)$ est un prédicat.

tout prédicat est une formule.

La condition d'appartenance d'une classe doit être compatible avec la structure du concept de cette classe. Une condition θ est compatible avec un concept cp (on le note $\theta \approx cp$) si elle satisfait les règles suivantes :

$$\frac{E \vdash p_1 \approx cp \quad E \vdash p_2 \approx cp}{E \vdash (p_1 \wedge p_2) \approx cp}$$

$$\frac{E \vdash cp.struct.a : D_i \quad v \in D_i \quad op : D_i \times D_i \longrightarrow \mathcal{B}}{E \vdash (a\; op\; v) \approx cp}$$

$$\frac{E \vdash cp.struct.a_1 : D_i \quad E \vdash cp.struct.a_2 : D_i \quad op : D_i \times D_i \longrightarrow \mathcal{B}}{E \vdash (a_1\, op\, a_2) \approx cp}$$

$$\frac{\begin{array}{l} E \vdash cp.struct.r : n_{cp} \quad E(n'_{cp}) = cp' \\ E \vdash cp'.struct.a : D_i \quad v \in D_i \quad op : D_i \times D_i \longrightarrow \mathcal{B} \end{array}}{E \vdash (r.val.a\, op\, v) \approx cp}$$

$$\frac{\begin{array}{l} E \vdash cp.struct.r_1 : n'_{cp} \quad E(n'_{cp}) = cp' \quad E \vdash cp'.struct.a_1 : D_i \\ E \vdash cp.struct.r_2 : n''_{cp} \quad E(n''_{cp}) = cp'' \quad E \vdash cp''.struct.a_2 : D_j \\ v \in D_i \quad op : D_i \times D_j \longrightarrow \mathcal{B} \end{array}}{E \vdash (r_1.val.a_1\, op\, r_2.val.a_2) \approx cp}$$

II.3.4 Taxonomies de spécialisation et classification

La classification d'instances consiste à calculer pour une instance donnée l'ensemble des classes du concept dont elle est une instance. On présente le résultat de la classification d'une instance par la minimisation de l'ensemble de ses classes par rapport à la relation de spécialisation. Nous reprenons la définition abstraite de la classification proposée par J. Euzenat [Euze93a, Euze93b] dans son étude des systèmes classificatoires.

Relation de spécialisation

◇ **Définition** *Spécialisation*: Une classe c_1 est une spécialisation d'une classe c_2 si c_1 et c_2 appartiennent au même concept et si sa condition d'appartenance θ_1 est une restriction de celle de c_2 : $(c_1 \leq c_2)$ *ssi* $(\theta_1(x) \Rightarrow \theta_2(x))$

La relation de spécialisation définit une relation d'ordre partiel sur l'ensemble $\mathcal{T} = cp.taxo$ des classes d'un concept. Si l'on impose à chaque concept d'admettre une plus grande classe pour la relation de spécialisation (la classe *racine* du concept), alors les classes d'un même concept peuvent être organisées en une taxonomie de spécialisation.

Un objet est rattaché à une seule classe, mais il peut appartenir à plusieurs classes :

◇ **Définition** *Instance*: Une instance o d'une classe c est un objet du concept qui satisfait sa condition d'appartenance : $c.\theta(o) = vrai$.

Un objet est instance de toutes les sur-classes de la classe à laquelle il est rattaché. Pour une classe c, $In(c)$ dénote l'ensemble de toutes ses instances et $At(c)$ l'ensemble de tous les objets qui lui sont rattachés. Pour un objet o, l'ensemble des classes dont il est une instance est noté $Cl(o)$. Cette distinction entre objet attaché à une classe et instance intervient dans les algorithmes d'évaluation de l'état courant de la base. En effet, seule la classe à laquelle un objet est rattaché est accessible aux algorithmes à partir de cet objet. Dans la suite nous utiliserons indifféremment *objet* et *instance*.

◇ **Définition** *Classe racine*: La classe racine d'un concept est le plus grand élément de sa taxonomie de spécialisation. Sa condition d'appartenance est *vrai*, donc tout objet du concept est une instance de cette classe. Elle est notée *cp.racine*.

◇ **Définition** *Taxonomie de spécialisation*: Une taxonomie de classes $\mathcal{T}$ est un ensemble de classes muni d'une relation d'ordre partiel, la relation de spécialisation notée $\leq$. La taxonomie des classes d'un concept *cp* est notée *cp.taxo*.

L'ensemble des instances d'une classe c peut être calculé à partir des ensembles d'objets attachés à ses sous-classes :

$$In(c) = \bigcup_{c' \leq c} At(c')$$

La relation de spécialisation est compatible avec la relation d'inclusion sur les classes :

$$c \leq c' \Longrightarrow In(c) \subseteq In(c')$$

Il est habituel de représenter une taxonomie sous la forme de sa réduction transitive, $c \rightarrow c'$ ssi :

$$(c \leq c') \wedge (\forall c''; ((c \leq c'') \wedge (c'' \leq c')) \Longrightarrow ((c'' = c) \vee (c'' = c')))$$

Classification d'instances

◇ **Définition** *Minimisation*: La minimisation d'un ensemble $\mathcal{S}$ par rapport à une relation d'ordre partiel sur ses éléments est l'ensemble :

$$\mu_R \mathcal{S} = \{x \in \mathcal{S}; \forall y \in \mathcal{S}, yRx \Rightarrow xRy\}$$

Ce sont les éléments minimaux de $\mathcal{S}$ pour R.

Si $\mathcal{T}$ est l'ensemble des classes du concept, et si $Cl(o) \subseteq \mathcal{T}$ est le sous-ensemble des classes dont o est instance, alors les *classes minimales* pour un objet o de ce concept sont donc :

$$\mu_{\leq} Cl(o) = \{c \in Cl(i); \forall c' \in Cl(o), c' \leq c \Rightarrow c' = c\}$$

Les classes terminales d'une instance o sont ses classes minimales qui n'admettent pas de sous-classes : $\nu Cl(o) = Cl(o) \cap \mu_{\leq}\mathcal{T}$ où $\mu_{\leq}\mathcal{T}$ est l'ensemble des puits de la taxonomie de classes $\mathcal{T}$.

Propriétés sémantiques de la classification

En fonction des contraintes imposées au modèle objet, plusieurs propriétés sont possibles pour la classification d'instances.
◇ **Définition** *Exhaustivité*: La classification d'instance est *exhaustive* si un objet qui appartient à une classe appartient nécessairement à l'une de ses sous-classes :

$$\forall c, (\exists c'; c' \leq c) \Longrightarrow (\forall o \in \mathcal{O}; c \in Cl(o), \exists c'; c' \leq c \wedge c' \in Cl(o))$$

La classification exhaustive a pour résultat de rattacher toute instance à au moins une classe terminale du concept.
◇ **Définition** *Exclusivité*: La classification est exclusive si une instance d'une classe ne peut être instance que d'au plus une seule de ses sous-classes :

$$\forall c, c', c''(c' \leq c) \wedge (c'' \leq c) \Longrightarrow$$
$$(\forall o \in \mathcal{O}, \forall c \in Cl(o), (c' \in Cl(o)) \wedge (c'' \in Cl(o)) \Rightarrow (c' \leq c'') \vee (c'' \leq c'))$$

◇ **Définition** *Convergence*: La classification est convergente si pour toute instance $o \in \mathcal{O}$, $Cl(o)$ admet un minimum :

$$\forall o \in \mathcal{O}, \exists c; \mu_{\leq} Cl(o) = \{c\}$$

Si la classification est exclusive ou convergente, alors elle a pour résultat de n'associer à une instance qu'une seule classe minimale :

$$\forall o \in \mathcal{O}; \mid \mu_{\leq} Cl(o) \mid = 1$$

Si la classification est exhaustive, alors pour toute instance, ses classes minimales sont des classes terminales :

$$\forall o \in \mathcal{O}; \mu_{\leq} Cl(o) = \nu Cl(o)$$

Complétude de la classification

Un objet modélise une entité de la résolution. Une classe modélise une catégorie d'entités qui partagent des propriétés. La classification est complète si elle détermine pour un objet donné toutes les classes dont il est instance. Le rattachement d'un objet à une classe dépend des informations associées à cet objet. Si elles sont complètes alors la classification est complète par rapport à la modélisation qui est faite de la résolution.

II.3.5 Base de données

◇ **Définition** *Base de données*: Une base de données bd est un couple $bd = (\mathcal{CP}, \mathcal{O})$ où $\mathcal{CP}$ est un ensemble fini de concepts et $\mathcal{O}$ un ensemble fini d'objets consistants avec $\mathcal{CP}$.

Le modèle objet suppose l'unicité des identificateurs d'objets, de classes et de concepts dans la base.

Environnement statique : schéma de la base

Les aspects statiques de la base constituent son schéma. Celui-là n'évolue pas pendant la résolution. Le schéma d'une base comprend la définition des concepts, de leurs classes, mais également celle des objets qui sont créés dans la base. N'est pas prise en compte l'appartenance des objets aux différentes classes ni leurs valeurs. Ces deux aspects sont gérés par l'environnement dynamique W.

L'environnement statique est noté E. Il permet en particulier d'accéder aux entités de la base à partir de leurs identificateurs. E peut donc être vu comme une fonction :

$$\begin{array}{rcl} ID \bigcup N_{Cp} \bigcup N_{Cl} & \longrightarrow & \mathcal{O} \bigcup \mathcal{CP} \bigcup \mathcal{CL} \\ id \in ID & \leadsto & o \in \mathcal{O} \\ n_{cp} \in N_{Cp} & \leadsto & cp \in \mathcal{CP} \\ n_{cl} \in N_{Cl} & \leadsto & cl \in \mathcal{CL} \end{array}$$

Environnement dynamique

Les actions élémentaires ont pour effet de réserver un objet dans un concept, d'affecter les valeurs des attributs d'un objet et de classer un objet. Ces opérations font donc intervenir un environnement dynamique W qui se décompose en trois sous-environnements : $W =< W_{Cp}, W_{Cl}, W_{Obj} >$. Par opposition à l'environnement statique, l'environnement dynamique évolue pendant la durée de la résolution.

Gestion des concepts Chaque concept est en fait un réservoir d'objets. Les objets disponibles dans un concept sont construits à la phase d'intialisation. Le nombre d'objets dans un concept est ainsi fixé pour toute la phase de résolution. Il est donc nécessaire de définir un mécanisme de gestion des objets pour chaque concept. Pour cela, on définit pour chaque concept cp les ensembles $Lib(cp)$ et $Alloc(cp)$ qui contiennent respectivement les objets libres et alloués du concept.

L'environnement dynamique $\mathcal{CP}$ qui rend compte des statuts des concepts est une fonction définie par :

$$\begin{array}{rrcl} W_{Cp} : & \mathcal{CP} & \longrightarrow & 2^{ID} \times 2^{ID} \\ & cp & \leadsto & < Lib(cp), Alloc(cp) > \end{array}$$

L'expression $W_{/cp \leadsto [O_1, O_2]}$ signifie que dans l'environnement W, le concept cp admet O_1 pour ensemble d'objets libres et O_2 pour ensemble d'objets alloués. L'ensemble des objets libres d'un concept est en fait une pile d'objets. On note $W_{/cp \leadsto [O_1.o,O_2]}$ pour indiquer que o est le premier objet disponible dans l'ensemble des objets libres du concept cp dans l'environnement dynamique W.

Gestion des classes La caractérisation locale de la base est faite au travers de l'appartenance des objets aux différentes classes. Les instances sont gérées par l'environnement dynamique :

$$\begin{array}{rrcl} W_{Cl}: & N_{Cl} & \longrightarrow & 2^{ID} \\ & n_{cl} & \leadsto & At(n_{cl}) \end{array}$$

Gestion des objets L'environnement dynamique W_{Obj} rend compte des affectations des valeurs des objets de la base :

$$\begin{array}{rrcl} W_{Obj}: & ID \times A & \longrightarrow & \mathcal{V} \\ & < id, a > & \leadsto & v = E(id).val.a \end{array}$$

Propriétés imposées au modèle utilisé

Nous nous plaçons dans le cas d'une base d'objets où **la classification est exclusive mais pas nécessairement exhaustive**. En effet, le modèle objet est conçu pour manipuler et classer des objets incomplets. Les mises à jour de la base ont pour but de compléter les valeurs des attributs de ces objets et de les faire migrer d'une classe vers une autre au moyen de la classification. La classification n'est alors plus complète, l'appartenance d'un objet à une classe peut en effet dépendre d'informations manquantes. Ces informations seront éventuellement fournies pendant la résolution, ce qui permettra de compléter le résultat de la classification.
Remarque : La définition de la classification telle qu'elle a été donnée ci-dessus ne spécifie pas le comportement d'un objet par rapport à la classification des objets avec lesquels il est en relation. Notre modèle n'a pas pour objectif de mettre en œuvre les objets composites [Mari90], bien qu'ils présentent un intérêt certain. Aussi, nous ne définissons pas la classification des composants d'un objet. C'est lors des mises à jour de la base qu'il faut spécifier la reclassification des objets composites qui sont concernés par ces mises à jour.

II.4 Actions et comportement des objets

Les mises à jour de la base se font par des actions composées qui sont des séquences prédéfinies d'actions élémentaires. Ces actions ont pour effet de modifier l'état des objets de la base.

II.4.1 Statut d'un objet

Lors des mises à jour de la base, les objets sont créés, modifiés ou supprimés. Par rapport à son utilisation possible, le statut d'un objet créé mais non valué est différent de celui d'un objet créé, valué et rattaché à une classe.

- Un objet d'un concept est *alloué* si son identificateur est retenu. Il est *libre* sinon.
- Un objet est *affecté* quand les valeurs de ses attributs sont (partiellement ou totalement) renseignées. Il est *vide* quand son contenu est non significatif.
- Un objet est *attaché* à une classe quand d'une part cette classe lui est associée comme la plus spécialisée, et quand d'autre part il fait partie des objets attachés à cette classe. Quand il n'est rattaché à aucune classe, un objet est *détaché*.

Le statut d'un objet se décompose en trois parties qui rendent respectivement compte de son statut libre ou alloué, valué ou vide, attaché ou détaché. Son statut sera donc donné par un triplet :

$$< L/A, o.val/-, \emptyset/o.cl > \in \{L, A\} \times \mathcal{V} \times \mathcal{CL}$$

Le premier élément indique si l'objet est libre (L) ou alloué (A), le second fournit sa valeur v s'il est affecté, $(-)$ sinon, le troisième indique la classe à laquelle il est rattaché, $\emptyset$ si il n'est pas rattaché.

Toutes les combinaisons ne sont pas licites. Pour être rattaché à une classe un objet doit vérifier la condition d'appartenance de cette classe. Il doit donc avoir été préalablement alloué et valué. Les valeurs légales du statut d'un objet sont données par le tableau suivant :

$< L, -, \emptyset >$
$< A, -, \emptyset >$
$< A, v, \emptyset >$
$< A, v, c >$

TAB. II.1 - *Tableau des statuts licites d'un objet.*

L'expression $W_{/o \leadsto s}$ avec $s \in \{L, A\} \times \mathcal{V} \times \mathcal{CL}$ signifie que l'objet o a le statut s dans l'environnement dynamique W.

II.4.2 Actions élémentaires

Les actions élémentaires agissent sur le statut des objets, elles consistent à allouer ou libérer un objet, affecter un objet, le rattacher ou le détacher d'une classe. Elles sont définies par les fonctions suivantes :

$$\begin{array}{lrcl}
allouer\colon & N_{Cp} & \longrightarrow & ID \\
lib\acute{e}rer\colon & ID & \longrightarrow & N_{Cp} \\
affecter\colon & ID \times A \times \mathcal{V} & \longrightarrow & ID \\
rattacher\colon & ID \times N_{Cl} & \longrightarrow & ID \\
d\acute{e}tacher\colon & ID & \longrightarrow & ID
\end{array}$$

Leur sémantique opérationnelle est donnée par les expressions suivantes :

allouer

$$\frac{E(n_{cp}) = cp}{\mathcal{A}(E)(W_{/cp \rightsquigarrow [O_1.o, O_2]})(allouer(n_{cp})) = (W_{/cp \rightsquigarrow [O_1, O_2 \cup \{o\}]\ o \rightsquigarrow <A,-,\emptyset>}\ , E(o))}$$

$$\frac{E(n_{cp}) = cp}{\mathcal{A}(E)(W_{/cp \rightsquigarrow [\emptyset, O]})(allouer(n_{cp})) = (W, Err)}$$

libérer

$$\frac{E \vdash o \approx cp \quad E(id) = o \quad E, W \vdash o \in Alloc(cp)}{\mathcal{A}(E)(W_{/cp \rightsquigarrow [O_1, O_2 \cup \{o\}]})(liberer(id)) = (W_{/\{cp \rightsquigarrow [O_1.o, O_2]\ o \rightsquigarrow <L,-,\emptyset>\ \}}, id)}$$

attacher

$$\frac{\begin{array}{l} E \vdash o \approx cp \quad E(id) = o \quad E(n_{cl}) = cl \quad E \vdash cl =< n_{cl}, p, n_{cp}, \theta > \\ E, W \vdash o \in Alloc(cp) \quad E, W \vdash o \rightsquigarrow < A, v, \emptyset > \quad E, W \vdash \theta(o) = vrai \end{array}}{\mathcal{A}(E)(W_{/O = At(cl)})(attacher(id, n_{cl}) = (W_{/\{At(cl) = O \cup o; o \rightsquigarrow <A,v,n_{cl}>\ \}}, id)}$$

détacher

$$\frac{\begin{array}{l} E \vdash o \approx cp \quad E(id) = o \quad E(n_{cp}) = cp \quad E(n_{cl}) = cl \quad E \vdash cl =< n_{cl}, p, n_{cp}, \theta > \\ E, W \vdash o \in Alloc(cp) \quad E, W \vdash o \in At(cl) \quad E, W \vdash o \rightsquigarrow < A, v, n_{cl} > \end{array}}{\mathcal{A}(E)(W_{/At(cl) = O \cup o})(detacher(id, n_{cl}) = (W_{/\{At(cl) = O; o \rightsquigarrow <A,v,\emptyset>\ \}}, id)}$$

affecter

$$\frac{E \vdash o \approx cp \quad E(id) = o \quad E(n_{cl}) = cl \quad E, W \vdash o \in Alloc(cp) \qquad E \vdash s = cp.struct \quad E \vdash s.a : \tau \quad E \vdash v : \tau}{\mathcal{A}(E)(W)(affecter(id, a, v) = (W_{/o.val.a=v;\}}, id)}$$

classif

L'action élémentaire *classif* détermine la classe la plus spécialisée à laquelle l'instance peut être rattachée, mais elle ne modifie pas l'environnement. Elle ne peut pas échouer, car une instance peut toujours être rattachée à la classe racine de son concept.

$$\frac{E(id) = o \quad E \vdash o \approx cp \quad E(n_{cl}) = cl \qquad E \vdash cl \in cp.taxo \quad E, W \vdash cl' = \mu_{\leq} cp.taxo(o)}{\mathcal{A}(E)(W)(classif(id, n_{cl})) = (W, E(cl'))}$$

La classification admet une classe comme paramètre pour initialiser la recherche. Un algorithme de classification dans un système classificatoire est constitué de deux parties [Mari93] : le parcours du graphe de spécialisation et l'appariement de l'objet classé avec les nœuds du graphe. Cet appariement dans notre modèle consiste à évaluer si l'objet satisfait la condition d'appartenance de la classe. L'algorithme de classification utilisé est le suivant :

```
classif2 (o,c)
début
    pour toute sous-classe c' de c
        si (c'.θ(o) == vrai)
          alors
              retourner classif2(o, c')
    retourner c
fin

classif (o,c)
début
    si (c.θ(o) == vrai)
      alors
          retourner classif2(o, c)
      sinon
          retourner classif2(o, c.cp.racine)
fin
```

Algorithme récursif de classification d'un objet o à partir d'une classe c dans un modèle objet où la classification est exclusive mais non exhaustive.

II.4.3 Actions composées

Les mises à jour de la base ne se font pas par les actions élémentaires mais par des actions composées qui constituent des séquences d'actions élémentaires. L'utilisation de ces séquences prédéfinies garantit la cohérence du contenu de la base.

classer

L'action composée *classer* rattache une instance à la classe la plus spécialisée du concept à laquelle elle appartient. Cette action est utilisée dans les autres actions composées, mais elle est mise à la disposition de l'utilisateur pour qu'il puisse spécifier la reclassification d'un objet dont il sait que des composants ont été modifiés.

Pour ne définir qu'une seule action composée *classer*, nous faisons intervenir la classe virtuelle $\top$ qui, pour tout concept, est une sur-classe de sa classe racine.

$$\frac{\begin{array}{c}\mathcal{A}(E)(W_1)(classif(id, n_{cl})) = (W_1, n'_{cl}) \\ \mathcal{A}(E)(W_1)(attacher(id, n'_{cl})) = (W_2, id)\end{array}}{\mathcal{A}(E)(W_1)(classer(id, n_{cl})) = (W_2, id)}$$

créer

L'action composée *créer* alloue un nouvel objet dans le concept, l'affecte et le rattache après l'avoir classé.

$$\frac{\begin{array}{l}E(n_{cp}) = cp \\ \mathcal{A}(E)(W_1)(allouer(n_{cp})) = (W_2, id) \\ E(id) = o \quad E, W_1 \vdash o \leadsto < A, -, \emptyset > \\ \mathcal{A}(E)(W_2)(affecter(id, a, v)) = (W_3, id) \\ \mathcal{A}(E)(W_3)(classer(id, \top)) = (W_4, id)\end{array}}{\mathcal{A}(E)(W_1)(creer(a, v)) = (W_4, id)}$$

supprimer

L'action composée *supprimer* détache l'objet concerné et le libère.

$$\frac{\begin{array}{l}E(id) = o \quad E, W_1 \vdash o \leadsto < A, v, n_{cl} > \\ E(n_{cl}) = cl \quad E \vdash o.cpt = n_{cp} \\ \mathcal{A}(E)(W_1)(detacher(id, n_{cl})) = (W_2, id) \\ \mathcal{A}(E)(W_2)(liberer(id)) = (W_3, id)\end{array}}{\mathcal{A}(E)(W_1)(supprimer(id)) = (W_3, id)}$$

modifier

L'action composée *modifier* opère sur un objet attaché à une classe. L'objet est détaché, affecté puis classé.

$$\frac{\begin{array}{l} E(id) = o \quad E, W_1 \vdash o \leadsto < A, v, n_{cl} > \\ \mathcal{A}(E)(W_1)(detacher(id, n_{cl})) = (W_2, id) \\ \mathcal{A}(E)(W_2)(affecter(id, a, v)) = (W_3, id) \\ \mathcal{A}(E)(W_3)(classer(id, n_{cl})) = (W_4, id) \end{array}}{\mathcal{A}(E)(W_1)(modifier(id, a, v)) = (W_4, id)}$$

Remarque : L'action composée *classer* telle qu'elle est utilisée dans les actions composées *créer* et *modifier* n'agit que sur un objet libre, mais affecté et qui était attaché à la classe à partir de laquelle la classification est activée. Cette remarque sera utilisée pour l'évaluation incrémentale de la base.

II.5 Caractérisation de l'état de la base

Le rôle principal de la base étant de caractériser l'état courant de la résolution, nous étudions dans cette section les mécanismes utilisés pour réaliser la caractérisation locale et globale. La caractérisation locale concerne chacun des objets de la base, la caractérisation globale concerne l'ensemble de la base, elle est définie à partir de la notion de configuration.

II.5.1 Caractérisation des objets

La caractérisation locale de la base est réalisée par le mécanisme de classification d'instances qui pour un objet détermine ses propriétés en fonction des classes dont il est instance (voir II.3.4).

II.5.2 Détection d'événements

La caractérisation globale du contenu de la base consiste à détecter les états particuliers de la base qui intéressent le contrôle de la résolution. Ces états particuliers définissent les événements associés au contenu de la base. La détection de ces événements est réalisée au moyen de filtres que nous appelons *configurations*. Les configurations permettent de détecter les événements qui correspondent à des situations particulières.

Evénements et configurations

Un événement est constitué d'un tuple d'instances qui satisfait une condition donnée. Cet événement peut intéresser le contrôle ou rendre activable une méthode. Une configuration est une catégorie d'événements qui sont définis par la même condition.

Une configuration est donc une condition qui porte sur un sous-ensemble d'instances de la base. Elle est composée d'une part d'un contexte et d'autre part d'une conjonction de prédicats. Le contexte définit les variables par rapport aux classes de la base. Les prédicats appliquent des tests aux variables du contexte.

Syntaxe des configurations

La syntaxe du langage de définition des configurations est donnée par la grammaire suivante :

<*configuration*>	::=	<*contexte*>$^+$: <*prédicat*>* ;
<*contexte*>	::=	<*classe*> vide
	|	existe <*variable*> dans <*classe*> ;
<*prédicat*>	::=	<*objval*> <*op*> <*valeur*>
	|	<*objval*> <*op*> <*objval*> ;
<*objval*>	::=	<*variable*>.val.<*attribut*> ;

Le contexte d'une configuration comprend, des exigences de classes vides, et des références à des instances de certaines classes. Le nombre de variables d'instances fixé par le contexte est le **rang** de la configuration.

Exemple de configuration

La présence d'une fissure large trop profonde pour que la pièce examinée soit acceptée est détectée par la configuration suivante :

CFG-fissure-larg-prof = *existe* x *dans* CL-fissure-large :
x.profondeur $\geq s_{fp}$

où la constante numérique s_{fp} spécifie le seuil de profondeur.

Calcul des configurations

Soit N_v un ensemble dénombrable de noms de variables. Le calcul des configurations est défini par les prédicats et les formules :

si $x \in N_v$, si $op : \mathcal{V} \times \mathcal{V} \rightarrow \mathcal{B}$ et si $v \in \mathcal{V}$ alors $(x.val.a \; op \; v)$ est un prédicat où la variable x est libre.

si $x_1, x_2 \in N_v$ et $op : \mathcal{V} \times \mathcal{V} \rightarrow \mathcal{B}$ alors $x_1.val.a_1 \; op \; x_2.val.a_2$ est un prédicat où les variables x_1 et x_2 sont libres.

si p_1 et p_2 sont des prédicats, alors $p_1 \wedge p_2$ est un prédicat dont les variables sont libres.

tout prédicat est une formule.

si la variable x est libre dans la formule ϕ et si $n_{cl} \in N_{Cl}$, alors $(\exists x \in n_{cl})(\phi)$ est une formule où la variable x est liée.

$\diamond$ **Définition** *Configuration*: Un configuration est une formule dont toutes les variables sont liées. Elle est notée $cfg = (\mathcal{X}, \theta)$ où $\mathcal{X}$ est le contexte et θ la conjonction de prédicats.

Correction des configurations

Les configurations correctes sont construites à partir des prédicats en vérifiant les règles suivantes :

$$\frac{\begin{array}{l} n_{cl} \in N_{Cl} \;\; E(n_{cl}) = c \;\; E \vdash c.struct = s \;\; E \vdash s.a = D_i \\ v \in D_i \;\; op : D_i \times D_i \rightarrow \mathcal{B} \end{array}}{E, \mathcal{X}.\{x : n_{cl}\} \vdash x.val.a \; op \; v : \mathcal{B}}$$

$$\frac{\begin{array}{l} n_1 \in N_{Cl} \;\; E(n_1) = c_1 \;\; E \vdash c_1.struct = s_1 \;\; E \vdash s_1.a_1 = D_i \\ n_2 \in N_{Cl} \;\; E(n_2) = c_2 \;\; E \vdash c_2.struct = s_2 \;\; E \vdash s_2.a_2 = D_j \\ op : D_i \times D_j \rightarrow \mathcal{B} \end{array}}{E, \mathcal{X}.\{x_1 : n_1, x_2 : n_2\} \vdash x_1.val.a_1 \; op \; x_2.val.a_2 \; : \; \mathcal{B}}$$

$\diamond$ **Définition** *Type d'un contexte*: Le type d'un contexte est le tuple de ses classes :

$$n_i \in N_{Cl} \;\; E(n_i) = c_i \;\; T(\{x_1 : n_1, ..., x_p : n_p\}) = (c_1, ..., c_p)$$

Les éléments d'un contexte sont donc les tuples d'instances qui appartiennent au produit cartésien des ensembles d'instances des classes qui le constituent :

$$In(T(\{x_1 : n_1, ..., x_p : n_p\})) = In(c_1) \times ... \times In(c_p)$$

Sémantique des configurations

Les éléments d'une configuration sont les éléments de son contexte qui vérifient sa condition.La sémantique dénotationnelle des configurations est donc donnée par :

$$\mathcal{E}_{E,W}[\![(\mathcal{X}, \theta)]\!] = \{o = (o_1, ...o_p) \in In(T(\mathcal{X})) \,/\, \theta(o) = vrai\}$$

Les algorithmes ne peuvent accéder qu'aux ensembles d'objets attachés aux classes. L'expression utilisée pour calculer les tuples qui appartiennent à une configuration est donc la suivante :

$$\mathcal{E}_{E,W}[\![(\mathcal{X},\theta)]\!] = \{o = (o_1, ...o_p) \in \prod_{i=1}^{p} \bigcup_{c'_i \leq c_i} At(c'_i) \,/\, \theta(o)\}$$

La sémantique d'une configuration peut donc s'exprimer par l'application de l'opérateur de sélection σ au produit cartésien des ensembles d'instances des classes qui apparaissent dans le contexte :

$$\mathcal{E}_{E,W}[\![(\mathcal{X},\theta)]\!] = \sigma_\theta(In(c_1) \times ... \times In(c_p))$$

Cette expression sera utilisée pour le calcul incrémental des configurations à partir des mises à jour de la base.

II.5.3 Configurations et classes

Les configurations ne sont pas indépendantes des classes, elles définissent de nouvelles classes dont les instances sont les tuples d'instances qui valident les configurations.

Contextes des configurations et concepts

Le contexte d'une configuration définit la structure d'un nouveau concept. On note $cp_\mathcal{X}$ le concept défini par le contexte $\mathcal{X}$ d'une configuration $(\mathcal{X},\theta)$. La structure de ce concept comprend les attributs relation définis par les variables du contexte : $a_j = \gamma(\mathcal{X}, x_i)$. Elle est notée $\Gamma(\mathcal{X})$

Le type d'un attribut a_j est le concept de la classe qui est associée à la variable correspondante dans le contexte :

$$\frac{\begin{array}{c}\mathcal{X} = (...x_i : n_i...) \;\; s = \Gamma(\mathcal{X}) \;\; a_i = \gamma(\mathcal{X}, x_i) \\ n_i \in N_{Cl} \;\; E(n_i) = c_i \;\; E \vdash c_i.cp = n_{cp}\end{array}}{E \vdash s.a_i : n_{cp}}$$

Le contexte d'une configuration spécifie l'appartenance des variables par rapport à des classes. Cette contrainte exprimée sur les variables définit une condition $\theta_\mathcal{X}$ d'appartenance à la classe qui correspond à la configuration :

$$\theta_\mathcal{X} = \bigwedge_{i=1}^{p} (\gamma(\mathcal{X}, x_i) \in n_i)$$

Calculs des configurations et des conditions

Le calcul des conditions des configurations tel qu'il a été défini, permet de spécifier les conditions d'appartenance de nouvelles classes. En appliquant la substitution $\sigma[x_i/\gamma(x_i)]$ pour toutes les variables x_i du contexte qui interviennent dans la condition de la configuration, on définit bien la condition θ_θ d'une classe.

Une configuration $cfg = (\mathcal{X}, \theta)$ définit donc un concept $cp_\mathcal{X}$ et une classe de ce concept dont la condition d'appartenance est $\theta_{cl} = \theta_\mathcal{X} \wedge \theta_\theta$.

Configurations et schéma de la base

La définition de configurations définit de nouveaux concepts et de nouvelles classes qui modifient le schéma de la base. Les contextes de deux configurations peuvent définir le même concept mais elles définissent deux classes différentes. De façon générale, un ensemble de configurations définit n_1 concepts et n_2 classes, avec $n_1 \leq n_2$. Les concepts communs à plusieurs configurations sont évidemment factorisés sous la forme d'un seul concept. Un ensemble de configurations définit ainsi plusieurs concepts et des ensembles de classes pour ces concepts. Il se pose alors le problème de la construction des graphes de spécialisation pour ces concepts. Ce problème correspond à la classification de classes.

Pour que les classes associées à deux configurations soient comparables pour la relation de spécialisation, il faut qu'elles appartiennent au même concept. L'égalité du rang de deux configurations ne suffit pas à les associer au même concept, il faut de plus qu'elles aient des contextes qui définissent le même concept. Les configurations ayant été définies indépendamment les unes des autres, les éventuels liens de spécialisation entre leurs classes doivent être calculés automatiquement si l'on souhaite exploiter la relation de spécialisation entre ces classes pour évaluer les configurations.

Catégorisation des configurations

La catégorisation est la construction d'un ensemble C de classes sous la forme d'une taxonomie de catégories [Euze93a] : il s'agit donc de classification de classes. Cette construction fait intervenir un critère de catégorisation $<<$ avec lequel est compatible la relation de spécialisation [Euze93b]. Selon le modèle objet utilisé, différents critères de catégorisation sont envisageables. Outre les travaux menés dans le cadre des langages terminologiques pour le calcul de la subsumption, une approche basée sur les types aborde le problème de la classification de classes [Chai89a] [Capp93]. Cette approche consiste à construire pour chaque attribut du concept, l'ensemble des types qui lui sont associés. A chaque classe est associé le type correspondant. Et dans le graphe des types d'un attribut, à chaque type correspond le sous-ensemble des classes du concept pour lesquelles il est associé à cet attribut. La localisation des types associés aux attributs de la nouvelle classe dans les graphes des types de ces attributs permet de localiser la classe dans le graphe de spécialisation.

Cette approche est opérationnelle dans le système Tropes [Mari90], nous ne traitons donc pas le problème de la classification des classes qui sont définies par les configurations. Nous considérons que la classification de classes est mise en œuvre par les mécanismes décrits ci-dessus.

Il n'est pas nécessaire de définir un nouvel environnement dynamique pour rendre compte du contenu des configurations. En effet, du point de vue du contenu de la base, les configurations sont des classes ordinaires, mise à part le fait que leur mise à jour est effectuée par les mécanismes de gestion de la base et non par des actions extérieures. Il est ainsi possible de définir des configurations qui portent sur d'autres configurations.

II.5.4 Algorithme de calcul

L'algorithme d'évaluation d'une configuration est celui du calcul d'une sélection appliquée à un produit cartésien. La relation de spécialisation étant une relation d'ordre partiel, une classe est sous-classe d'elle même.

Pour une configuration $(\mathcal{X}, \theta)$, avec $T(\mathcal{X}) = In(c_1) \times ... \times In(c_n)$, l'algorithme est le suivant :

```
calcul-cfg(< c1, ..., cn >,θ)
début
    pour toute classe c'1 sous-classe de c1
        pour tout objet o1 attaché a c'1
            ...
                pour toute classe c'n sous-classe de cn
                    pour tout objet on attaché a c'n
                        si θ(o1, ..., on) = vrai
                            alors insérer o = (o1, ..., on) dans cfg
fin
```

Algorithme général de calcul d'une configuration

De nombreux travaux ont été consacrés à l'étude du calcul de jointures dans les bases de données relationnelles. Un aperçu des techniques proposées est donné par Mishra [Mish92]. Pour optimiser ce calcul, les classes sont ordonnées par cardinalité croissante. La complexité de ce type de calcul [Mish92] est : $O(m_1 \times ... \times m_n)$ où m_i est le nombre d'instances de la classe c_i. Si m_i^j est le nombre d'objets attachés à la $j$ème sous-classe de c_i qui en compte q_i, alors le nombre de tests est : $O((m_1^1 + ... + m_1^{q_1}) \times ... \times (m_n^1 + ... + m_n^{q_n}) = \prod_{i=1}^{n}(\sum_{j=1}^{q_i} m_i^j))$.

Les bases de données orientées objets intègrent peu à peu ces techniques. Mais aucune (à ce jour!) ne met en œuvre le mécanisme de classification d'instances. Les travaux publiés dans la littérature ne sont pas directement applicables pour le calcul incrémental des configurations dans notre modèle objet.

L'algorithme général ci-dessus est valable pour une configuration dont le rang est connu et égal à n. Pour calculer les ensembles de tuples des configurations de rang variable, il faut définir un algorithme dans lequel le rang des configurations est un paramètre.

```
rjoin (tuple,Tclasses,rang,i)
début
    si (i == rang)
       alors
            tester(cfg, tuple)
       sinon
            c_i ← Tclasse[i]
            pour toute sous-classe c'_i de c_i
                  pour tout objet o attaché a c'_i
                        tuple[i] ← o
                        rjoin(tuple, Tclasse, rang, (i + 1))
fin

rjoin([−, ..., −], [c_1, ..., c_n], r, 0)
```

Algorithme récursif de calcul d'une configuration. La fonction tester *teste l'appartenance du tuple à la configuration, et en cas de succès l'insère dans son ensemble de tuples.*

II.6 Evaluation incrémentale des configurations

Les mises à jour de la base sont effectuées au moyen des actions composées. Elles sont donc bien identifiées et concernent chacune peu d'instances. Il est ainsi intéressant de ne pas recalculer les configurations mais de calculer seulement les modifications incrémentales de leurs ensembles de tuples.

II.6.1 Approches utilisées pour l'évaluation incrémentale

Le calcul incrémental des configurations consiste à mettre à jour leurs ensembles de tuples d'instances directement après chaque modification du contenu de la base. D'un point de vue abstrait, la recherche d'éléments qui vérifient une condition donnée relève d'une problématique commune à plusieurs domaines de l'informatique :

- Les techniques de compilation dans les systèmes de production sont utilisées pour optimiser la phase d'évaluation des règles activables. Elles comprennent deux étapes : une étape d'unification des faits modifiés avec chacun des filtres des règles et une étape de jointure entre les ensembles de faits obtenus lors de l'unification. Une sémantique des systèmes de production est proposée par F. Fagès [Fage92].

- Que ce soit pour les bases de données déductives ou actives, les mécanismes d'évaluation des règles activables associent les techniques de compilation développées dans les systèmes de production aux techniques d'optimisation physique des bases de données [Jeus92] [Abit92]. Ces techniques dépendent fortement du modèle de données utilisé et du langage de requêtes associé [Bert92] : relationnel et Datalog [Fabr92], objet [Stra90] [Jeus92]. Les modèles objets font apparaître la notion d'ensemble pour les classes d'objets [Pfal88] [Pfal92], ce qui motive des travaux de recherche sur l'intégration des ensembles dans les langages de requêtes [Bals91] [Chen91] [Rund92] [Qian91]. Les algorithmes proposés ne prennent pas (encore) en compte la classification d'instances, mais certains résultats sont intéressants dans cette perspective.

- Dans un système de représentation par objets, les filtres sont les opérations qui consistent à rechercher toutes les instances de la base qui vérifient un modèle appelé filtre [Napo92]. SHIRKA ([Rech85]) définit des filtres et les évaluent avec l'algorithme RETE. Le système CLASSIC [Borg89b] de la famille KL-ONE [Brac85] met en œuvre les filtres dans un langage terminologique. Le calcul des filtres exploite la relation de subsumption entre les filtres pour éviter de tester une instance avec des filtres qui sont comparables.

- Un problème de satisfaction de contraintes (CSP) est défini par la donnée de n variables et de leurs domaines respectifs ainsi que de p contraintes qui portent sur ces variables. Les algorithmes [Kuma92] développés pour la résolution d'un CSP (recherche d'une solution, de toutes les solutions) consistent soit à générer les tuples de valeurs qui satisfont les contraintes, soit à propager les contraintes sur les domaines des variables, soit à combiner ces deux méthodes. La génération des tuples de valeurs qui satisfont les contraintes du CSP est plus complexe que l'évaluation des configurations. En particulier, l'aspect combinatoire est beaucoup plus critique.

L'unification d'un fait avec le filtre d'une règle de production correspond au test d'appartenance d'un objet avec une classe. Cependant, lors de sa création, un objet est directement spécifié par rapport à sa classe (la classe racine de son concept en général). Après sa création, les migrations d'un objet sont dirigées par le graphe de spécialisation. Il n'y a donc pas lieu de chercher à optimiser l'arbre d'unification. Par contre, la notion de réseau qui à une classe associe toutes les configurations dans lesquelles elle intervient, est pertinente. Il serait de plus possible d'optimiser l'appariement d'un objet avec une classe [Mari93].

Calcul incrémental dans les bases de données

Le calcul incrémental des relations dans une base de données relationnelle a été étudié par X. Qian [Qian91]. Elle propose un algorithme de calcul qui prend en compte les opérateurs de sélection, de produit cartésien, de jointure et de projection. Nous reprenons cette approche pour prouver nos algorithmes de calcul des configurations en introduisant la relation de spécialisation.

Pour un ensemble R, l'ensemble des éléments qui sont ajoutés lors d'une mise à jour est noté $\triangle R$, celui des éléments qui sont supprimés est noté $\bigtriangledown R$. Afin de calculer les modifications incrémentales des ensembles, les opérateurs suivants sont définis :

union disjointe : l'expression $R \oplus S$ est égale à $R \cup S$ si R et S sont disjoints, sinon elle n'est pas définie.

différence contenue : l'expression $R \ominus S$ est égale à la différence ordinaire $R - S$ si S est inclus dans R, sinon elle n'est pas définie.

En utilisant ces deux opérateurs, l'ensemble R après modification (noté R^{nouv}) s'exprime en fonction de son contenu avant modification (noté R^{anc}) :

$$R^{nouv} = (R^{anc} \oplus \triangle R) \ominus \bigtriangledown R = (R^{anc} \ominus \bigtriangledown R) \oplus \triangle R$$

Pour calculer $\triangle R$ et $\bigtriangledown R$ lorsque R est une expression composée (cas d'une configuration) on utilise les règles de propagation suivantes :

sélection (σ_θ) :

$\sigma_\theta(R \oplus S) \Longrightarrow \sigma_\theta(R) \oplus \sigma_\theta(S)$
$\sigma_\theta(R \ominus S) \Longrightarrow \sigma_\theta(R) \ominus \sigma_\theta(S)$

produit cartésien :

$(R \oplus S) \times T \Longrightarrow (R \times T) \oplus (S \times T)$
$(R \ominus S) \times T \Longrightarrow (R \times T) \ominus (S \times T)$

X. Qian montre que les règles de propagation préservent l'équivalence : si $\mathcal{E} \Longrightarrow \mathcal{F}$ est une règle de propagation et que $\mathcal{E}$ est défini, alors $\mathcal{F}$ est défini et $\mathcal{E} = \mathcal{F}$.

Si en appliquant une ou plusieurs règles de propagation, l'expression $\mathcal{F}$ peut se déduire de $\mathcal{E}$, alors $\mathcal{E}$ peut se transformer en $\mathcal{F}$, ce qui se note : $\mathcal{E} \Longrightarrow^* \mathcal{F}$

En appliquant successivement les règles de propagation à une expression de modification, il est possible d'obtenir une expression dont toutes les mises à jour sont isolées :

$$\mathcal{E}[R/R \oplus \triangle R] \Longrightarrow^* \mathcal{E} \oplus \triangle \mathcal{E}$$

$$\mathcal{E}[R/R \ominus \bigtriangledown R] \Longrightarrow^* \mathcal{E} \ominus \bigtriangledown \mathcal{E}$$

X. Qian démontre cette propriété par induction sur la structure de $\mathcal{E}$.

II.6.2 Actions et calcul incrémental

En appliquant les résultats précédents au calcul des configurations, il est possible de déterminer l'influence d'une action sur les configurations. Soient cfg^{nouv} et cfg^{anc} les ensembles de tuples d'instance d'une configuration avant et après mises à jour de la base.

Si les mises à jour correspondent à l'ajout de nouvelles instances dans la classe c_i, le calcul d'une configuration qui comprend cette classe est obtenu en appliquant les règles de propagation :

$$cfg = \sigma_\theta(In(c_1) \times ... \times In(c_i) \times ... \times In(c_n))$$
$$In^{nouv}(c_i) = In^{anc}(c_i) \oplus \triangle In(c_i)$$
$$cfg^{nouv} = \sigma_\theta(In(c_1) \times ... \times (In^{anc}(c_i) \oplus \triangle In(c_i)) \times ... \times In(c_n))$$
$$cfg^{nouv} \Longrightarrow^* cfg^{anc} \oplus \sigma_\theta(In(c_1) \times ... \times \triangle In(c_i) \times ... \times In(c_n))$$
$$\triangle cfg = \sigma_\theta(In(c_1) \times ... \times \triangle In(c_i) \times ... \times In(c_n))$$

L'ensemble $\triangle In(c_i)$ des nouvelles instances de la classe c_i contient toutes les instances qui ont été rattachées aux sous-classes de c_i :

$$\triangle In(c_i) = \bigcup_{c \leq c_i} \triangle At(c)$$

Le rattachement de nouvelles instances aux sous-classes de la classe c_i a pour effet sur une configuration qui comprend c_i d'ajouter les tuples qui correspondent à l'expression suivante :

$$\triangle cfg = \sigma_\theta(In(c_1) \times ... \times (\bigcup_{c \leq c_i} \triangle At(c)) \times ... \times In(c_n))$$

Les modifications incrémentales d'une configuration pour l'ajout et le retrait d'instances dans l'une des classes qui composent son contexte, sont données par :

$$\triangle cfg = \bigcup_{c \leq c_i} \sigma_\theta(In(c_1) \times ... \times \triangle At(c) \times ... \times In(c_n))$$

$$\bigtriangledown cfg = \bigcup_{c \leq c_i} \sigma_\theta(In(c_1) \times ... \times \bigtriangledown At(c) \times ... \times In(c_n))$$

II.6.3 Algorithmes d'évaluation incrémentale

Cette section présente les algorithmes de recalcul incrémental des configurations respectivement pour les actions élémentaires, pour les actions composées et pour les mises à jour générales.

Ces algorithmes utilisent deux fonctions :

- La fonction $caltuples(cfg, pos, id)$ calcule les tuples créés par un objet id en position pos dans la configuration cfg : $\sigma_\theta(In(c_1) \times ...\{id\}... \times In(c_n))$
- La fonction $rechtuples(cfg, pos, id)$ retourne la liste des tuples de la configuration cfg qui comportent l'objet id en position pos.

Calcul incrémental pour une action élémentaire

Les expressions précédentes permettent d'évaluer les modifications dues aux opérations élémentaires *attacher* et *détacher*. Les algorithmes qui calculent ces modifications incrémentales sont donc les suivants :

```
△ajout-cfg(id, c0)
début
    pour toute classe c ≥ c0
        pour toute configuration cfg telle que
        cfg = σθ(In(c1) × ... × In(ci) × ... × In(cn)) avec ci = c
            △cfg ← caltuples(cfg,i,id)
fin
```

Calcul des nouveaux tuples pour les configurations qui sont concernées par l'action élémentaire attacher(id,c).

```
▽enlever-cfg(id, c0)
début
    pour toute classe c ≥ c0
        pour toute configuration cfg telle que
        cfg = σθ(In(c1) × ... × In(ci) × ... × In(cn)) avec ci = c
            ▽cfg ← rechtuples(cfg,i,id)
fin
```

Calcul des tuples à supprimer pour les configurations qui sont concernées par l'action élémentaire détacher(id,c).

Calcul incrémental pour une action composée

Lors des mises à jour de la base, les actions élémentaires ne sont utilisées que dans les actions composées. Il est donc possible d'exploiter le contexte de déclenchement d'une action élémentaire pour optimiser le calcul incrémental des configurations.

L'action élémentaire *attacher* est utilisée dans l'action composée *classer*. Cependant, les effets du rattachement de l'objet classé à une classe c dépendent du statut de cet objet avant la classification :

- Si l'objet id était libre, alors l'action $attacher(id, c)$ affecte toutes les sur-classes de c.

- Si l'objet id était rattaché à une classe c_1 sur-classe de c_2, alors l'action $attacher(id, c_2)$ affecte toutes les classes c telles que $c_1 \geq c \geq c_2$.

- Si l'objet id était rattaché à une classe c_1 sous-classe de c_2, alors l'action $attacher(id, c_2)$ affecte toutes les classes c telles que $c_1 \leq c \leq c_2$.

Pour prendre en compte la classe à laquelle était rattachée l'instance dans l'action composée $classer(id, n_{cl})$, on définit les fonctions $ajoutcfg\downarrow(id, c_1, c_2)$ et $enlevercfg\uparrow(id, c_1, c_2)$.

Les algorithmes de recalcul incrémental des configurations sont alors :

$\triangle ajoutcfg\downarrow(id, c_1, c_2)$
début
  pour toute classe c telle que $c_1 > c \geq c_2$
    pour toute configuration cfg telle que
    $cfg = \sigma_\theta(In(c'_1) \times ... \times In(c_i) \times ... \times In(c'_n))$ avec $c_i = c$
      $\triangle cfg \leftarrow caltuples(cfg, i, id)$
fin

Calcul des nouveaux tuples pour les configurations qui sont concernées par l'action élémentaire $ajoutcfg\downarrow$.

$\triangledown enlevercfg\uparrow(id, c_1, c_2)$
début
  pour toute classe c telle que $c_1 \leq c < c_2$
    pour toute configuration cfg telle que
    $cfg = \sigma_\theta(In(c'_1) \times ... \times In(c_i) \times ... \times In(c'_n))$ avec $c_i = c$
      $\triangledown cfg \leftarrow rechtuples(cfg, i, id)$
fin

Calcul des tuples à supprimer pour les configurations qui sont concernées par l'action élémentaire $enlevercfg\uparrow$.

Les actions composées sont alors redéfinies pour intégrer l'étape d'évaluation des configurations :

classer*

L'action composée *classer** agit sur les configurations qui sont concernées par les classes qui sont placées entre la classe c_1 d'origine et la classe c_2 de destination :

$$\frac{\begin{array}{l}\mathcal{A}(E)(W_1)(classif(id, n_{c_1})) = (W_1, n_{c_2}) \\ E(n_{c_1}) = c_1 \quad E(n_{c_2}) = c_2 \quad E \vdash c_1 > c_2 \\ \mathcal{A}(E)(W_1)(ajoutcfg\downarrow(id, c_1, c_2)) = (W_2, id) \\ \mathcal{A}(E)(W_2)(attacher(id, c_2)) = (W_3, id)\end{array}}{\mathcal{A}(E)(W_1)(classer^*(id, n_{c_1})) = (W_3, id)}$$

$$\frac{\begin{array}{l}\mathcal{A}(E)(W_1)(classif(id, n_{c_1})) = (W_1, n_{c_1})\\ E(n_{c1}) = c_1 \ \ E(n_{c_2}) = c_2 \ \ E \vdash c_1 < c_2\\ \mathcal{A}(E)(W_1)(enlevercfg\!\uparrow(id, c_1, c_2)) = (W_2, id)\\ \mathcal{A}(E)(W_1)(attacher(id, c_2)) = (W_3, id)\end{array}}{\mathcal{A}(E)(W_1)(classer^*(id, n_{c_1})) = (W_3, id)}$$

$$\frac{\begin{array}{l}\mathcal{A}(E)(W_1)(classif(id, n_{c_1})) = (W_1, n_{c_1})\\ E(n_{c1}) = c_1 \ \ E(n_{c_2}) = c_2 \ \ E \vdash c_1 = c_2\\ \mathcal{A}(E)(W_1)(attacher(id, c_2)) = (W_2, id)\end{array}}{\mathcal{A}(E)(W_1)(classer^*(id, n_{c_1})) = (W_2, id)}$$

créer*

L'action composée *creer** n'agit sur les configurations que par l'intermédiaire de l'action composée *classer**:

$$\frac{\begin{array}{l}E(n_{cp}) = cp \ \ \mathcal{A}(E)(W_1)(allouer(n_{cp})) = (W_2, id)\\ E(id) = o \ \ E, W_1 \vdash o \leadsto < A, -, \emptyset >\\ \mathcal{A}(E)(W_2)(affecter(id, a, v)) = (W_3, id)\\ \mathcal{A}(E)(W_3)(classer^*(id, \top)) = (W_4, id)\end{array}}{\mathcal{A}(E)(W_1)(creer^*(id, a, v)) = (W_4, id)}$$

supprimer*

L'action composée *supprimer** agit sur les configurations qui sont concernées par toutes les sur-classes de la classe dont on supprime l'élément :

$$\frac{\begin{array}{l}E(id) = o \ \ E, W_1 \vdash o \leadsto < A, v, n_{cl} >\\ E(n_{cl}) = cl \ \ E \vdash o.cpt = n_{cp}\\ \mathcal{A}(E)(W_1)(detacher(id, n_{cl})) = (W_2, id)\\ \mathcal{A}(E)(W_2)(enlevercfg\!\uparrow(id, n_{cl}, \top)) = (W_3, id)\\ \mathcal{A}(E)(W_3)(liberer(id)) = (W_4, id)\end{array}}{\mathcal{A}(E)(W_1)(supprimer^*(id)) = (W_4, id)}$$

modifier*

L'action composée *modifier** n'agit sur les configurations que par l'intermédiaire de l'action composée *classer**:

$$\frac{\begin{array}{c} E(id) = o \quad E, W_1 \vdash o \rightsquigarrow < A, v, n_{cl} > \\ \mathcal{A}(E)(W_1)(detacher(id, n_{cl})) = (W_2, id) \\ \mathcal{A}(E)(W_2)(affecter(id, a, v)) = (W_3, id) \\ \mathcal{A}(E)(W_3)(classer^*(id, n_{cl})) = (W_4, id) \end{array}}{\mathcal{A}(E)(W_1)(modifier^*(id, a, v)) = (W_4, id)}$$

Réseau des configurations

Dans les algorithmes proposés pour évaluer les effets des actions composées sur les configurations, les boucles d'accès aux configurations n'explicitent pas l'ordre dans lequel les configurations sont examinées.

Nous reprenons la notion de réseau des techniques de compilation en associant à chaque classe le réseau de toutes les configurations pour lesquelles elle apparaît dans le contexte. Nous avons vu en II.5.3 que l'ensemble des configurations définit un ensemble de concepts et de nouvelles classes, et que pour les configurations qui définissent le même concept, les classes qu'elles définissent peuvent être comparées par la relation de spécialisation. Il est donc intéressant d'exploiter cette propriété pour ne pas tester un tuple d'instances avec des configurations qui sont comparables.

Si la classification d'instances permet d'exploiter le graphe de spécialisation pour tester l'appartenance d'un objet aux différentes classes d'un concept, la classification telle que nous l'avons définie (II.3.4) est exclusive : un objet ne peut pas être instance de deux sous-classes directes d'une même classe. Il n'y a donc pas de multi-spécialisation. Or la construction automatique du graphe de spécialisation sur les classes des configurations qui définissent le même concept ne garantit pas l'absence de multi-spécialisation. Au contraire, il est fortement probable que les conditions associées aux différentes configurations soient telles que cette propriété ne soit pas vérifiée.

Il est donc nécessaire de mettre en œuvre un nouvel algorithme de classification [Mari93] pour prendre en compte le problème de la multi-spécialisation dans le graphe de spécialisation des classes associées aux configurations.

II.7 Conclusion : évaluation du modèle objet

Le modèle objet présenté dans ce chapitre combine classifications d'instances et détection d'événements pour caractériser l'état courant de la base. Il est donc bien adapté pour modéliser la résolution dans un système à base de connaissances réactif.

En proposant nos algorithmes d'évaluation incrémentale des configurations, notre objectif n'est pas de rechercher les méthodes optimales, mais seulement de montrer comment le mécanisme de classification d'instances influence les techniques d'évaluation. Nos algorithmes peuvent évidemment être améliorés. Cependant, même avec des algorithmes optimaux, certaines limites de ce modèle soulèvent des problèmes par rapport aux exigences d'un système réactif :

1. La définition des configurations pour gérer les événements est très coûteuse en place mémoire et peut poser des problèmes de débordement. Même en renforçant les conditions des configurations pour limiter le nombre de leurs éléments, il faut alors mettre en œuvre des techniques d'oubli qui consistent à supprimer certains des éléments d'une configuration [Euze91]. Cet oubli se fait en appliquant des critères pour supprimer les éléments les plus anciens par exemple. Ces techniques non monotones requièrent un système de maintien du raisonnement (TMS) pour répercuter les effets des suppressions des différents éléments.

2. L'évaluation des configurations est coûteuse en temps. Si une mise à jour de la base agit sur de nombreuses instances, l'évaluation du nouvel état de la base peut être prohibitive en temps. Or cette évaluation a pour rôle de détecter les événements de la résolution afin que le système leur apporte éventuellement une réponse. La qualité d'une réponse décroît avec le temps qui s'écoule. Donc si l'évaluation de la base est trop longue en temps, son intérêt est diminué pour le comportement réactif du système. Il faut donc envisager de mettre en œuvre une évaluation rationnelle (voir I.3.4) de l'état courant de la base.

3. Un système réactif comme TRAM permet d'envisager le parallélisme de l'exécution des actions qu'il pilote. Dans ce contexte, il faut alors mettre en œuvre des transactions concurrentes qui mettent chacune à jour la base et gérer les conflits entre ces transactions et le mécanisme d'évaluation de la base.

Ces différentes directions de recherche montrent bien l'intérêt des systèmes de production basés sur la classification et plus particulièrement en ce qui concerne les systèmes de production d'objets.

Chapitre III

Sélection dynamique de méthodes

Ce chapitre étudie le problème de la sélection dynamique de méthodes dans un $\mathcal{ERP}$ réactif [Chai93]. L'approche considérée est basée sur les notions de tâches et de méthodes. Les solutions proposées mettent en œuvre le mécanisme de classification d'instances dans une représentation par objets. Le modèle objet qui est utilisé pour mettre en œuvre ces algorithmes est celui qui est présenté dans le chapitre II. Toutefois, dans un souci de généralité, ce chapitre ne fait pas référence aux caractéristiques de ce modèle, hormis les deux dernières approches qui utilisent les configurations. Mises à part la classification et la gestion des configurations, l'approche adoptée pour la sélection dynamique de méthodes dans une architecture de contrôle réactif se veut indépendante du modèle objet utilisé.

Nous étudions tout d'abord (section III.1) le rôle des tâches et des méthodes pour la planification en résolution de problèmes. Puis nous posons le problème de la sélection dynamique de méthodes dans un $\mathcal{ERP}$ (section III.2). Enfin nous proposons quatre approches pour aborder ce problème :

1. La première approche (section III.3) associe à chaque tâche la taxonomie de sélection qui organise l'ensemble de ses méthodes en fonction de leurs conditions d'activabilité et de leur utilité. La sélection est faite par

classification d'une instance qui contient les données d'exécution.

2. La deuxième approche (section III.4) redéfinit l'algorithme de classification pour maîtriser le temps de délibération afin d'obtenir un mécanisme de sélection rationnelle d'instances.

3. La troisième approche (section III.5) consiste à exploiter les possibilités d'activation pour définir le mode de contrôle opportuniste. Nous utilisons les configurations pour gérer les événements qui correspondent aux possibilités d'activation des méthodes d'une tâche.

4. L'approche mixte (section III.6) combine la sélection par classification avec la gestion des possibilités d'activation.

III.1 Modélisation du contrôle par les tâches

Cette section se préoccupe de l'utilisation des tâches pour modéliser l'aspect dynamique des connaissances de résolution dans un $\mathcal{ERP}$. Elle ne rend pas compte des aspects méthodologiques liés à l'obtention de cette modélisation. Pour l'étude des méthodologies en acquisition des connaissances, nous renvoyons à [Dien90] et [Ermi93].

III.1.1 Tâches dans les systèmes à base de connaissances

La notion de *tâche* intervient dans plusieurs domaines de l'informatique, elle est essentiellement associée aux processus qui traitent les événements dans un système temps réel. Dans les systèmes à base de connaissances, la tâche identifie les buts et sous-buts pour modéliser les connaissances de contrôle en résolution de problèmes. Cependant, même parmi les systèmes à base de connaissances, plusieurs définitions des tâches sont utilisées.

A. Newell : Problématique de la modélisation - 1981

Plusieurs solutions ont été successivement proposées pour modéliser le contrôle en résolution de problèmes. Les systèmes à base de règles mettent en œuvre l'idée selon laquelle les connaissances de contrôle doivent être séparées des connaissances propres au domaine. Dans ces systèmes, le contrôle est spécifique par rapport au problème traité. L'étape suivante a donc consisté à modéliser les connaissances stratégiques. Par opposition aux approches qui faisaient référence aux représentations des connaissances (règles ou frames), l'approche de A. Newell dans sa modélisation en plusieurs niveaux (*Knowledge Level* [Newe82]) sépare la modélisation des connaissances de leur représentation. D'autres approches sont envisageables pour décrire un système de façon abstraite : la spécification abstraite d'algorithmes et les niveaux de D. Marr (*modèle, algorithme, réalisation* [Marr82]).

Le *niveau connaissance* d'A. Newell constitue une étape importante dans la modélisation des connaissances. Les niveaux identifiés par A. Newell dans un système informatique sont les suivants : physique (matériaux), composants (transistors), circuits électriques (portes ET, OU), circuits logiques (transfert de registres), symbolique (langages de programmation), connaissance. Chaque niveau admet une définition *structurelle* en fonction de la couche inférieure et une définition *fonctionnelle* interne à lui même.

Le *niveau connaissance* est exprimé dans le langage de programmation du niveau symbolique et définit le comportement du système par rapport à un *principe de rationalité* selon lequel les actions sont sélectionnées de manière optimale pour atteindre les buts de l'agent. Cet agent rationnel est composé de *connaissances*, d'un *ensemble d'actions* et d'un *ensemble de buts*. Les *lois de comportement* régissent l'utilisation des actions pour réaliser les buts.

Bien qu'ayant ses limites (en particulier le niveau connaissance ne peut être totalement défini sans faire référence au support informatique : sa définition fonctionnelle est donc incomplète [Gana91]), ce modèle a servi de point de départ aux différents travaux sur la modélisation et l'acquisition des connaissances qui sont basés sur les tâches. La notion de tâche étant introduite pour décrire la réalisation d'un but indépendamment du langage de représentation, plusieurs approches ont exploré la modélisation des principes de résolution à partir des tâches en associant à cette notion des définitions et des niveaux d'abstraction différents [Karb90] [Tann91]. La plupart des travaux relatifs aux tâches sont liés soit à l'acquisition des connaissances soit à l'explication. Nous présentons ci-dessous cinq travaux qui ont contribué à imposer les tâches en modélisation des connaissances, et dont les résultats apparaissent (explicitement ou non) dans les approches actuelles.

B. Chandrasekaran (1) : Tâches Génériques - 1983

B. Chandrasekaran [Chan86] s'est attaché à identifier les tâches abstraites en résolution de problèmes et à expliciter les connaissances relatives à la résolution d'une tâche. Une Tâche Générique organise et représente les connaissances nécessaires à la réalisation d'une catégorie de problèmes. Elle est caractérisée par ses entrées et ses sorties. Plusieurs tâches génériques ont été dénombrées et mises en œuvre dans le système MDX. Pour les problèmes de diagnostic, ce sont : la classification hiérarchique, la comparaison d'hypothèses, l'accès aux informations et l'assemblage abductif. En conception, ce sont la conception hiérarchique par sélection de plans et l'abstraction d'un état.

Parallèlement, W. Clancey a décrit une tâche abstraite sous le nom de Classification Heuristique [Clan85]. Cette tâche de diagnostic est constituée de trois étapes : un modèle abstrait du problème posé est construit; ce modèle abstrait est mis en correspondance avec les modèles abstraits de solutions connues; à partir du modèle de la solution obtenu, la solution du problème initial est construite. Ce mécanisme constitue une illustration importante de l'approche des tâches génériques.

J. Breuker & B. Wielenga: KADS - 1987

KADS (Knoweldge Acquisition and Design System) est une méthodologie pour l'acquisition des connaissances lors du développement d'un système à base de connaissances [Breu85] [Wiel92]. Elle a été conçue dans le cadre d'un projet ESPRIT. Le cycle de vie d'un système à base de connaissances comprend un modèle *conceptuel*, un modèle *fonctionnel* et un modèle de *conception*. Le langage de modélisation conceptuelle (KCML) organise les connaissances en quatre niveaux:

- Le niveau *domaine* contient les connaissances statiques du domaine en termes de concepts et de relations.
- Le niveau *inférence* décrit les compétences fonctionnelles qui permettent de faire des pas élémentaires d'inférence, comme les choix, les calculs.
- Le niveau *tâche* fournit les décompositions des tâches en sous-tâches pour contrôler les inférences afin de réaliser un but.
- Le niveau *stratégie* définit le contrôle associé aux tâches. C'est lui qui spécifie en fonction du contexte quelles tâches doivent être réalisées pour résoudre un problème.

Des langages de modélisation et des outils de développement sont proposés autour de la méthode KADS. Elle se veut donc exhaustive, et concerne par nature les éléments de bas niveau dans la modélisation des connaissances de résolution.

S. Marcus & J. McDermott: Méthodes à rôles limités - 1987

Le projet mené par J. McDermott porte sur la conception d'une plateforme de programmation assistée (Spark) dont les objectifs et les caractéristiques sont proches de ceux des $\mathcal{ERP}$. Il met en application les résultats des travaux qu'il a conduits en acquisition des connaissances.

L'approche des *méthodes à rôles limités* [McDe88] se préoccupe de la caractérisation des méthodes par rapport au domaine. Dans ce contexte, une méthode de résolution de problèmes désigne les actions qui permettent de résoudre un problème. Elle utilise des “mécanismes” pour sélectionner les actions à déclencher. Une *méthode à rôle limité* est la spécialisation d'une méthode associée à une tâche donnée.

Le système Spark [Klin91] est basé sur l'utilisation et la réutilisation de *mécanismes* pour réaliser les tâches de la résolution d'un problème. Cette approche repose sur le postulat qu'un développeur sait exprimer la définition d'une tâche en termes spécifiques du domaine, mais ne connait pas nécessairement les moyens de réaliser cette tâche. A partir de la description du problème, le système peut identifier les configurations de mécanismes possibles et sélectionner, en fonction du contexte, celle qu'il va réellement appliquer.

M. Musen & E. Shortliffe: Méthodes, Tâches et Mécanismes - 1987

L'équipe de M. Musen [Puer93] travaille sur la conception d'outils pour l'acquisition des connaissances dans les applications médicales. Les systèmes développés mettent en œuvre les différentes approches qui ont été successivement proposées.

PROTEGE et PROTEGE-II sont les deux générations d'outils indépendants du domaine qui constituent les supports des systèmes liés aux applications.

Le premier met en œuvre l'approche du *raffinement de squelettes de plans* qui consiste à instancier un plan prédéfini et à rechercher, pour chaque sous-tâche du plan, la méthode qui permet de la réaliser dans le contexte courant de la résolution. Cette utilisation évolutive de plans existants a été largement reprise dans les systèmes qui opèrent en milieu dynamique afin de pallier les limites de la planification statique.

PROTEGE-II reprend la même approche en formalisant plus précisément les relations entre les tâches, les méthodes et les mécanismes. Une méthode décrit la décomposition d'une tâche en sous-tâches, et une tâche élémentaire est réalisée par un mécanisme.

B. Chandrasekaran (2): Structure de Tâche - 1992

La structure de tâche est proposée par B. Chandrasekaran [Chan92] comme approche unificatrice des travaux précédents. Elle définit précisément les notions de tâche et de méthode.

Tâche : une tâche décrit une catégorie de problèmes (par exemple, le diagnostic).

Méthode : une méthode est un moyen pour réaliser une tâche donnée.

A une tâche peuvent être associées plusieurs méthodes parmi lesquelles une seule sera effectivement *sélectionnée* pour résoudre un problème donné. Une méthode peut être *calculatoire* (action physique ou module algorithmique), elle peut définir la décomposition de la tâche en sous-tâches (plan d'actions) ou elle peut fournir des opérateurs qui opèrent sur l'espace des sous-tâches pour identifier ou construire la décomposition de la tâche principale. La construction de la décomposition constitue la génération de plans.

Ces définitions sont compatibles avec les tâches génériques d'une part et avec l'exploration de l'espace des solutions d'autre part. Une Tâche Générique est effectivement un type de problèmes (diagnostic, conception) qui est indépendant du domaine et qui peut être utilisé pour construire différents systèmes à base de connaissances. L'utilisation d'opérateurs pour explorer l'espace des solutions permet de construire de nouvelles séquences de sous-tâches pour réaliser la tâche principale. Ce type de méthodes a été mis en avant par A. Newell dans l'architecture SOAR [Lair87].

La structure de tâche supprime donc la limite arbitraire qui distinguait les tâches génériques des autres tâches. Elle reconnait ainsi la relativité de la généricité d'une tâche, et renvoie son évaluation à l'analyse du système développé. Ainsi une tâche comme la conception peut être primitive dans certains problèmes et au contraire décomposée en sous-tâches dans d'autres problèmes. Cette approche est donc compatible avec celles qui s'intéressent aux éléments de bas niveau (KADS et les opérateurs de construction de modèles de Clancey [Clan92]) et avec celles qui abordent les éléments de plus haut niveau (tâches génériques).

Le rôle de la tâche pour la modélisation du contrôle en résolution de problèmes étant posé, il importe de disposer d'une représentation des tâches adaptée aux besoins des $\mathcal{ERP}$.

III.1.2 Représentation et gestion des tâches dans un $\mathcal{ERP}$

Le concept de tâche étant pris au sens de la structure de tâche, plusieurs représentations sont envisageables pour les tâches.

Tâches et $\mathcal{ERP}$

Le rôle des $\mathcal{ERP}$ étant de piloter des éléments de solutions (plans ou modules algorithmiques), c'est la phase de sélection dynamique de méthodes qui est importante dans le contrôle. La génération de plans peut être intéressante, mais elle n'est pas indispensable dans la mesure où le rôle des $\mathcal{ERP}$ n'est pas de construire des solutions pour des problèmes non résolus [Poll92]. Par contre, l'évaluation de la résolution par rapport aux différents éléments de solutions disponibles constitue un objectif prioritaire : le rôle de chaque plan, de chaque module algorithmique doit pouvoir être positionné par rapport à la résolution qui a été ou qui aurait pu être menée. Nous verrons que l'une des manières d'atteindre cet objectif est de mettre en œuvre la gestion des possibilités d'activation (voir I.2.7).

En résumé, la modélisation du contrôle par les tâches dans un $\mathcal{ERP}$ met l'accent sur la sélection dynamique de méthodes.

Le système OCAPI

OCAPI est un $\mathcal{ERP}$ dédié à la vision [Clem93, Thon93]. Ce système est bien représentatif des différents éléments qui interviennent dans le contrôle de la résolution pour gérer la planification et l'exécution de modules algorithmiques. OCAPI fonctionne à partir de *requêtes* qui spécifient le *but* à atteindre, son *contexte* (les arguments) et les *contraintes* imposées aux arguments et aux résultats. A chaque but est associé un ensemble d'*opérateurs* qui définissent les possibilités de l'atteindre. Des *méthodes* permettent de choisir un opérateur pour réaliser un but donné. Enfin, des règles sont mises en œuvre pour prendre différentes décisions : - les règles de choix sélectionnent une méthode parmi celles qui sont associées à un but - les règles d'évaluation indiquent comment évaluer les résultats d'exécution des opérateurs - les règles d'initialisation gèrent les valeurs des paramètres d'exécution - les règles d'ajustement corrigent les valeurs des paramètres en fonction des précédentes exécutions de l'opérateur. Un moteur définit le cycle de résolution à partir de la décomposition d'un but en sous-buts et des éléments associés.

Choix d'une représentation des tâches

Parmi les représentations possibles des tâches, nous avons choisi celle de B. Rousseau [Rous88] qui est définie dans le système de représentation par objets Shirka [Rech85]. Cette approche est d'une part compatible avec la structure de tâche, et permet d'autre part, d'intégrer dans la modélisation le comportement des modules algorithmiques qui sont pilotés. Elle est donc particulièrement bien adaptée pour piloter des modules algorithmiques en informatique scientifique [Rech91] [Rech92]. Elle propose un mécanisme puissant de sélection dynamique de méthodes à partir du mécanisme de classification d'instances. Nous donnons ci-dessous une description succinte de cette approche, une description formelle étant proposée en III.3.

Modélisation d'une bibliothèque de modules algorithmiques La bibliothèque de modules algorithmiques est partitionnée en fonction des traitements que réalisent les différents modules. Une classe de cette partition constitue une tâche. Ainsi tous les modules qui permettent d'inverser une matrice sont associés à la tâche d'inversion de matrice. Ces modules correspondent à des formats différents ou utilisent des propriétés particulières de la matrice pour appliquer un algorithme spécifique. Les modules associés à une tâche constituent les méthodes qui permettent de la réaliser. On dispose alors d'un ensemble fini de méthodes pour réaliser les tâches qui sont dites élémentaires.

De même, les plans d'actions qui constituent des séquences prédéfinies de tâches deviennent les méthodes associées aux tâches composites. Ainsi à une tâche composite correspondent plusieurs décompositions en sous-tâches ou plans d'actions (Figure III.1). Par exemple, la tâche de résolution d'un problème d'équations aux dérivées partielles admet au moins deux décompositions en sous-tâches : celle de la méthode des différences finies et celle de la méthode des éléments finis [Bara90].

L'utilisation de cette modélisation pour la résolution de problèmes consiste à rechercher pour chaque tâche, la méthode la plus pertinente qui permet de la réaliser. Cette approche est un cas particulier de la planification hiérarchique [Regn90] et nous l'appelons *"sélection hiérarchique"* de méthodes.

Représentation par objets Chaque tâche est représentée par un concept dont les attributs décrivent ses entrées et ses sorties. Les méthodes qui sont associées

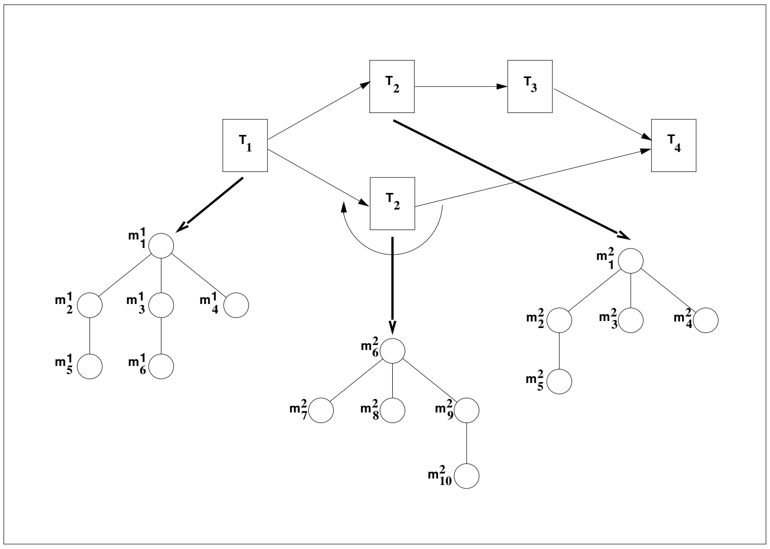

Fig. III.1 - *Sélection hiérarchique de méthodes : à chaque sous-tâche de la décomposition d'une tâche composite correspond une taxonomie de méthodes qui peut varier suivant le contexte de la sous-tâche.*

à cette tâche sont représentées par des classes. La relation de spécialisation qui définit les liens de sous-classes permet d'organiser les classes en fonction de la spécificité des méthodes qui leur sont associées. La notion de spécificité correspond aux conditions relatives au contexte d'exécution et qui, si elles sont satisfaites, permettent d'utiliser l'une ou l'autre des méthodes. Nous proposons une formalisation de cette notion dans la section III.3.

Lors de l'exécution d'une tâche, une instance est créée dans le concept qui lui correspond. Les attributs de cette instance sont affectés avec les données d'exécution de la tâche. La sélection de la méthode la plus spécifique est réalisée par le mécanisme de classification d'instances qui pour un objet détermine les classes du concept dont il est instance. Parmi celles-ci, la classe la plus spécialisée identifie la méthode la plus spécifique qui est compatible avec les données de la résolution. Ce mécanisme est étendu aux tâches composites, en associant aux classes du graphe de spécialisation du concept de la tâche, les plans d'actions qui constituent les décompositions de cette tâche en sous-tâches.

Réalisations & applications Cette approche a été mise en œuvre au sein du projet SHERPA dans les systèmes suivants : EVE, un $\mathcal{ERP}$ dédié aux équations aux dérivées partielles, et les systèmes génériques SCAI et SCARP. Les applications de cette approche sont résumées dans le tableau III.1.

Les $\mathcal{ERP}$ qui interviennent dans ces applications sont des systèmes coopératifs : ils nécessitent l'intervention de l'utilisateur pour sélectionner la tâche principale à exécuter. Nous ne détaillons pas plus la représentation par objets des tâches et des méthodes, ni les aspects propres à la coopération, ils font l'objet d'un travail de thèse [Will93]

Domaine	$\mathcal{ERP}$	Partenaires	Références
Equations aux dérivées partielles	EVE	LMC-IMAG	[Chai89b] [Bara90]
Traitement du signal	SCAI	IFREMER	
Analyse de données	SCARP	U. Claude Bernard	[Chev93]
Valeurs propres	SCAI	IRISA	
Contrôle commande	SCARP	ENSIEG	
Biologie moléculaire	SCARP	U. Claude Bernard Institut Pasteur	[Méd93]

TAB. III.1 - Applications du projet SHERPA qui utilisent l'approche objets pour modéliser une bibliothèque de modules algorithmiques

III.2 Problème

Nous nous intéressons au problème de la sélection dynamique de méthodes dans un $\mathcal{ERP}$ réactif. Il s'agit de piloter des solutions (plans d'actions ou modules algorithmiques) pour résoudre un problème. Ce travail n'aborde pas le problème de la génération de plans, mais seulement celui de la sélection.

III.2.1 Actions, tâches et méthodes

Une architecture de contrôle dispose d'un ensemble d'actions dont elle contrôle l'exécution pour réaliser le ou les buts qui lui sont assignés. Dans le cas d'un $\mathcal{ERP}$, les buts sont modélisés par des tâches et les actions sont des modules algorithmiques. Le rôle du système est de déterminer le moyen de résoudre la tâche courante à chaque étape de la résolution.

T. Tyrrell a étudié les mécanismes de sélection dynamique d'actions par rapport à plusieurs objectifs, et l'architecture qu'il propose permet de tester par simulation les différentes solutions [Tyrr93]. Ce travail ne prend pas en compte les notions de tâches et de méthodes. Selon l'approche de la structure de tâches [Chan92], les moyens qui permettent de réaliser une tâche donnée sont appelés les méthodes associées à cette tâche. Une tâche décomposable en sous-tâches est une tâche composite, sinon il s'agit d'une tâche élémentaire. Les méthodes associées à une tâche composite sont les plans d'actions qui définissent ses différentes décompositions en sous-tâches. Une tâche élémentaire n'est réalisable par le système que s'il existe une action qui permet de la réaliser. Les méthodes associées à une tâche élémentaire sont donc des actions.

III.2.2 Conflit de méthodes

Le problème de la sélection dynamique de méthodes dans une architecture de contrôle intervient lorsque plusieurs méthodes (plans ou actions) sont envisageables pour réaliser la tâche courante. Ce conflit entre les plans associés à une tâche composite ou les actions associées à une tâche élémentaire fait intervenir deux dimensions : l'activabilité des méthodes et leur intérêt pour la résolution.

- L'activabilité d'une méthode dépend de la situation courante : en fonction des données de la résolution, il est licite ou non d'utiliser cette méthode.
- L'intérêt d'une méthode dépend de ses performances et des principes de résolution : selon la phase courante de la résolution, une méthode rapide sera plus ou moins intéressante qu'une méthode précise.

Pour modéliser le conflit de méthodes, nous introduisons deux notions : la *condition d'activation* qui spécifie l'ensemble des états de la résolution qui autorisent l'utilisation de la méthode et l'*utilité* qui mesure l'intérêt d'une méthode par rapport à la résolution.

III.2.3 Sélection et classification

Le problème de la sélection dynamique de méthodes dans une architecture de contrôle est donc de déterminer la méthode activable la plus utile. Pour organiser les méthodes associées à une tâche, nous utilisons des relations d'ordre partiel :

- L'activabilité des méthodes est spécifiée par leur condition d'activation. En définissant une relation d'ordre partiel sur les conditions d'activation, on ordonne les méthodes par rapport à leur activabilité.
- L'utilité définit une priorité sur les méthodes.

La modélisation d'éléments partiellement ordonnés se fait naturellement par des taxonomies. Les taxonomies sont présentes dans les représentations par objets sous la forme de taxonomies de spécialisation. Il est donc intéressant d'utiliser une représentation par objets pour représenter l'ensemble des méthodes associées à une tâche.

Le mécanisme de classification d'instances permet d'exploiter la taxonomie de spécialisation pour déterminer les classes les plus spécialisées auxquelles l'instance classée peut être rattachée. La classification est donc bien adaptée pour mettre en œuvre le mécanisme de sélection de méthodes.

III.2.4 Classification et décision

Le mécanisme de classification d'instances est un mécanisme d'inférence : en caractérisant l'instance classée, il apporte de nouvelles informations relatives à l'état de la résolution. La sélection dynamique de méthodes est un problème de décision : les méthodes associées à la tâche courante constituent les différentes alternatives parmi lesquelles le système doit choisir celle qu'il active.

Rôle de la classification La classification d'instances consiste à déterminer pour un objet, toutes les classes dont il est instance. L'appartenance d'un objet à une classe donnée n'est pas l'objectif recherché, en fait le rôle de la classification est de caractériser l'objet classé pour en déduire des propriétés utiles à la prise décision.

Mesure compatible avec la classification Les propriétés déduites de la classification peuvent être exprimées sous la forme d'attributs attachés aux classes, mais ces attributs ne doivent pas être déterminants pour la classification, pour cela, ils ne doivent pas intervenir dans les conditions d'appartenance des classes.

Nous nous intéressons aux propriétés numériques munies d'une relation d'ordre total. Pour être compatible avec la classification d'instances, une mesure doit être monotone par rapport à la relation de spécialisation : une mesure u définie sur l'ensemble C des classes d'un concept et à valeurs dans un ensemble $(\mathcal{U}, \leq_u)$ est compatible avec la classification si, quelles que soient les classes c_1 et c_2 de C telles que $c_1 \leq c_2$, on a $u(c_1) \leq_u u(c_2)$.

III.3 Sélection hiérarchique de méthodes

Cette approche est une extension de la sélection hiérarchique de méthodes qui a été proposée par B. Rousseau [Rous88]. Il a utilisé la structure de classe dans une représentation par objets pour modéliser le contexte d'exécution dans lequel une méthode doit être sélectionnée. Notre approche introduit l'activabilité et l'utilité des méthodes : la condition d'activabilité d'une méthode spécifie les états de la résolution qui permettent de l'utiliser; l'utilité d'une méthode, à partir d'un état qui la rend activable, mesure son intérêt pour la résolution.

III.3.1 Modélisation d'une méthode

La modélisation du comportement d'une méthode est basée sur le contexte d'activation qui regroupe les variables dont les valeurs interviennent pour décider si la méthode est activable. Elles comprennent les entrées de la méthode (les paramètres d'appel), éventuellement les sorties (résultats) et d'autres données qui peuvent conditionner son comportement sans être directement prises en compte dans l'exécution de la méthode. La condition d'activation d'une méthode est l'ensemble des contraintes que doit satisfaire l'état courant de la résolution pour que cette méthode soit effectivement applicable. Le déclenchement d'une méthode dans un contexte d'exécution qui ne satisfait pas sa condition d'activation peut aboutir à un échec de l'exécution ou à des résultats non significatifs.

La modélisation du comportement d'une méthode m est composée de trois éléments :

- Le contexte d'activation $\mathcal{X}_m$ est un vecteur de variables qui permettent de spécifier le comportement de la méthode par rapport à son activabilité et à son utilité. Parmi ces variables, figurent ses entrées : $\mathcal{X}_m = [x_1 : \tau_1, ..., x_n : \tau_n]$ où x_i est le nom de la variable et τ_i son type.

- La condition d'activation θ_m est une conjonction de prédicats qui portent sur les variables du contexte d'activation. Il faut et il suffit que cette condition soit satisfaite par l'état courant de la résolution pour que la méthode soit activable : $\theta_m = \wedge_{i=1}^{q} p_i(x_1, ..., x_n)$. La condition d'activation d'une méthode décrit essentiellement des contraintes qui portent sur les valeurs de ses entrées. Une condition θ est évaluable sur un contexte $\mathcal{X}$ (on le note $\theta : \mathcal{X}$) si toutes les variables qui apparaissent dans θ sont définies dans $\mathcal{X}$.

- La partie action d'une méthode est, soit le plan d'actions, soit le module algorithmique qui met en œuvre la méthode. Les résultats de cette méthode peuvent être utilisés pour modifier les valeurs des variables du contexte d'activation.

Remarque : si la significativité des résultats d'un module algorithmique ne peut être décidée qu'après son exécution, comme c'est souvent le cas en traitement du signal, alors le déclenchement de ce module est considéré comme valide. Les tâches d'analyse de ces résultats vont intervenir dans la suite du plan d'actions pour décider de leur significativité.

III.3.2 Organisation de l'ensemble des méthodes

Les méthodes sont tout d'abord organisées en utilisant les relations qui existent entre leurs conditions d'activation.

Partition de l'ensemble des méthodes

L'ensemble des méthodes disponibles (plans et actions) est partitionné en fonction des tâches qu'elles réalisent. A chaque tâche T correspond ainsi le sous-ensemble M_T des méthodes qui permettent de la réaliser.

Par exemple, dans le cas d'une bibliothèque de calcul matriciel, à la tâche d'inversion de matrice correspondent tous les modules algorithmiques qui permettent d'inverser une matrice. Ces modules peuvent n'être applicables que sur des matrices qui présentent des propriétés particulières (symétrique, définie, positive), et ils mettent en œuvre des algorithmes différents qui exploitent ces propriétés.

Contexte d'activation d'une tâche

Le contexte d'activation $\mathcal{X}_T$ d'une tâche T regroupe toutes les variables qui interviennent dans les contextes d'activation de ses méthodes. Ainsi, les conditions d'activation des méthodes associées à la tâche T peuvent être évaluées à partir du contexte $\mathcal{X}_T : \mathcal{X}_T / \forall m \in M_T,\ \theta_m : \mathcal{X}_T$

L'ensemble des conditions qui peuvent être évaluées sur le contexte d'activation d'une tâche T est noté Θ_T.

Relation d'ordre partiel sur les condition d'activation

L'ensemble des conditions d'activation des méthodes associées à une tâche T est noté Θ_T^a. Une relation d'ordre partiel est définie sur l'ensemble des conditions d'activation des méthodes associées à une tâche T. Une condition d'activation θ_1 est une restriction de θ_2 si l'ensemble des états de la résolution qui satisfont θ_1 est inclus dans celui des états qui satisfont θ_2 :

$$\theta_1 \leq_\theta \theta_2 \Longleftrightarrow (\forall x \in \mathcal{X}_T, \theta_1(x) \Longrightarrow \theta_2(x))$$

Pour cette relation d'ordre partiel, plus une condition d'activation est restrictive, plus elle est petite. Cette relation d'ordre partiel définie par cette relation de restriction est une relation de subsumption (relation de préordre définie sur des éléments structurés à partir des relations d'ordre partiel définies sur les composants de ces structures). Les conditions d'activation étant définies sur des vecteurs de variables sans introduire de nouveaux concepts, la relation de subsumption définie ci-dessus est anti-symétrique, donc constitue bien une relation d'ordre partiel.

De nombreux travaux ont été menés pour calculer la relation de subsumption entre des conjonctions de prédicats appliqués à des termes logiques (voir II.1.2). Les résultats de ces travaux portent sur la complexité des algorithmes proposés pour le calcul de la subsumption. Cette complexité dépend des opérateurs utilisés pour construire les termes structurés.

Dans ce travail, nous considérons comme acquis le résultat de la construction du graphe de subsumption des méthodes associées aux tâches (voir discussion en II.5.3).

Hiérarchie des conditions d'activation d'une tâche

L'ensemble des conditions d'activation des méthodes associées à une tâche peut être organisée sous la forme d'une hiérarchie de restriction. L'ensemble M_T des méthodes associées à la tâche T est organisé par les conditions d'activation de ses éléments. Le graphe d'activation d'une tâche T est le graphe dirigé acyclique dont les nœuds sont les conditions d'activation et à chaque nœud correspond le sous-ensemble des méthodes dont il est la condition d'activation. La fonction $\mathcal{A}_T$ permet d'utiliser ce graphe :

$$\begin{array}{rccl} \mathcal{A}_T : & \Theta_T^a & \longrightarrow & 2^{M_T} \\ & \theta & \longmapsto & \{m \in M_T / \theta_m = \theta\} \end{array}$$

Nous supposons dans cette partie de la présentation que le graphe des conditions d'activation des méthodes associées à une tâche est une hiérarchie, appelée hiérarchie d'activation de la tâche. Cette restriction est forte, mais acceptable pour l'ensemble des méthodes qui interviennent par rapport à un seul critère de contrôle (voir III.3.6).

Exemple

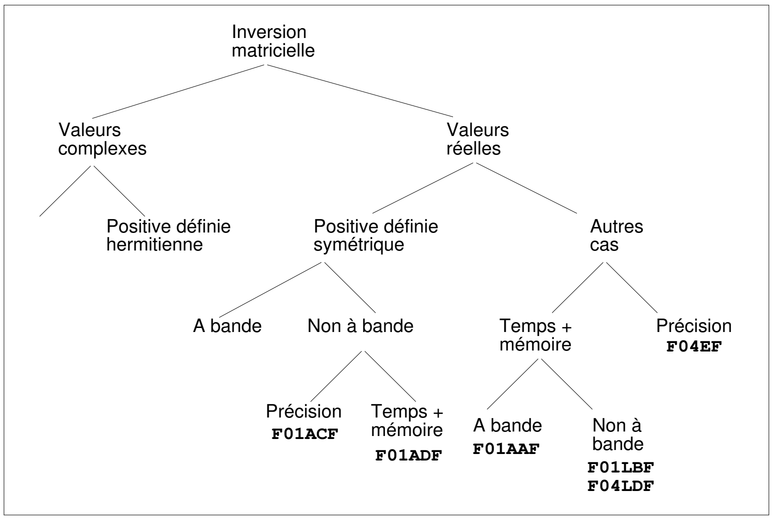

FIG. III.2 - *Hiérarchie de spécialisation des méthodes d'inversion de matrices. Les procédures utilisées font partie de la bibliothèque NAG.*

III.3.3 Modélisation par objets

Cette modélisation du comportement des méthodes associées à une tâche peut être mise en œuvre dans un système de représentation par objets.

Concept d'activation d'une tâche

Le contexte d'activation d'une tâche est modélisé par un concept. Les attributs de ce concept sont définis par les variables du contexte d'activation. Les classes qui décrivent le concept correspondent aux conditions d'activation des méthodes. Chaque condition d'activation définit ainsi la condition d'appartenance d'une classe du concept. La hiérarchie d'activation de la tâche est ainsi concrétisée par la hiérarchie de spécialisation du concept qui modélise le contexte d'activation de la tâche.

Limites liées au modèle objet

La limitation de cette approche réside dans la puissance d'expression du modèle objet utilisé pour représenter les concepts, les classes et les conditions d'activation. Si les conditions d'activation utilisent des requêtes qui ne peuvent pas s'exprimer dans le modèle objet, alors la méthode concernée ne pourra pas être incluse dans la modélisation. Nous faisons ici référence au modèle objet défini dans le chapitre précédent, mais le principe de cette approche reste valable pour d'autres modèles qui mettent en œuvre la classification d'instances.

III.3.4 Evaluation des possibilités par classification d'instances

L'utilisation d'un système de représentation par objets permet de rendre opérationnelle la modélisation des méthodes pour évaluer à partir de l'état courant de la résolution les méthodes activables.

Instanciation de la tâche

Dans un système de représentation par objets, la classification d'instances détermine pour une instance donnée les classes auxquelles elle appartient. Lorsque la tâche T doit être réalisée, une instance est créée dans son concept d'activation. Les attributs de cette instance sont affectés par les valeurs des variables du contexte d'activation. L'instance est alors classée. En déterminant les classes dont cette instance est élément, le mécanisme de classification d'instances analyse la situation. L'évaluation des possibilités d'activation est obtenue en récupérant les méthodes dont les classes résultats modélisent la condition d'activation.

Obtention des méthodes activables

Le schéma de la base étant fixé, seul son contenu W intervient pendant la classification. Si $\mathcal{C}_T$ est l'ensemble des classes du concept d'activation de la tâche T, et si $classif(W)(x) \subseteq \mathcal{C}_T$ est le résultat de la classification de l'instance x dont les valeurs d'attributs correspondent au contexte d'exécution, alors l'ensemble $M^a_T(W)(x)$ des méthodes activables pour la tâche T est donné par :

$$M^a_T(W)(x) = \bigcup_{c \in classif(W)(x)} \mathcal{A}_T(c.\theta)$$

où $c.\theta$ est la condition d'appartenance de la classe c (notation définie en II.3.3).

Obtention des méthodes activables les plus contraintes

Les méthodes les plus contraintes sont celles dont les conditions d'activation sont les plus restreintes, elles correspondent donc aux classes les plus spécialisées du concept d'activation. Si $\mu_{\leq}classif(W)(x)$ est l'ensemble des classes les plus spécialisées dont x est instance, alors l'ensemble $M_T^{+a}(W)(x)$ des méthodes activables les plus contraintes est donné par l'expression :

$$M_T^{+a}(W)(x) = \bigcup_{c \in \mu_{\leq}classif(W)(x)} \mathcal{A}_T(c.\theta)$$

On souhaite utiliser la relation de subsumption des conditions d'activation pour identifier la ou les méthodes les plus performantes. En effet, si on suppose que les méthodes les plus contraintes sont les plus performantes, alors l'ensemble des méthodes sélectionnées correspond à $M_T^{+a}(W)(x)$. Mais il est nécessaire d'affiner ce résultat, en explicitant la notion de performance par *l'utilité d'une méthode*.

III.3.5 Sélection par classification d'instances

La modélisation du comportement des méthodes associées à une tâche est enrichie afin de pouvoir sélectionner la méthode activable la plus performante. Pour cela, nous introduisons l'utilité d'une méthode et les conditions de sélection qui spécifient les états qui permettent d'exprimer cette utilité. Il est alors possible d'utiliser la classification d'instances pour sélectionner une méthode parmi celles qui sont associées à la tâche qui doit être réalisée.

Utilité d'une méthode

La résolution du conflit de méthodes consiste à sélectionner la méthode activable la plus performante. Si l'état courant de la résolution influe sur l'activabilité des méthodes, il influe aussi sur leur performance. Pour quantifier cette influence, on utilise la notion d'utilité. Si w est un état de la résolution, on note $\bar{U}_T([m, w])$ l'utilité d'exécuter la méthode m pour réaliser la tâche T à partir de l'état w. L'utilité est à valeur dans l'ensemble $\mathcal{U}$ muni d'une relation d'ordre total $\preceq$.

Cette notation reprend celle de Russell [Russ91] en introduisant la référence explicite à la tâche T, alors que Russell ne définit l'utilité d'une méthode que de façon absolue. Les travaux d'O. Etzioni portent sur le conflit d'actions par rapport à plusieurs buts et consiste à essayer les méthodes les unes après les autres [Etzi89]. Notre problème concerne le conflit d'actions par rapport à une tâche qui doit être réalisée. Le choix de la méthode associée à cette tâche est fixé en fonction de l'utilité des méthodes qui sont activables dans l'état courant de la résolution. L'utilité d'une méthode est relative à des critères de contrôle, mais dans cette section, elle n'est considérée que par rapport à un seul critère, elle est donc une mesure absolue. Les critères de contrôle seront pris en compte dans la sélection multi-critères.

Notre approche de la sélection dynamique de méthode est dédiée aux architectures de contrôle embarquées. Pour ce type de système, l'erreur d'exécution n'est pas acceptable. Les méthodes ne sont donc pas essayées les unes après les autres jusqu'à la réalisation de la tâche courante. Par conséquent, si une méthode n'est pas activable dans un état de la résolution w, son exécution à partir de cet état ne peut pas être envisagée, donc son utilité à partir de cet état est égale à $-\infty$.

Utilité d'une méthode pour un ensemble d'états

Nous supposons que le concepteur du système de décision est capable d'évaluer l'utilité d'une méthode pour des sous-ensembles d'états qu'il spécifie par des conditions. Si $\mathcal{W}$ est l'ensemble de tous les états possibles, W_θ est le sous-ensemble des états qui vérifient la condition θ: $W_\theta = \{w \in \mathcal{W} / \theta(w)\}$

L'utilité d'une méthode pour le sous-ensemble d'états W_θ est, par définition, égale à son utilité minimale sur cet ensemble. En effet, dans la perspective de l'absence d'échec, l'utilité sur un sous-ensemble d'états est l'utilité garantie quel que soit l'élément de ce sous-ensemble. Il s'agit donc bien du minimum des utilités associées à ces éléments.

On définit ainsi la fonction U_T qui, pour une méthode et pour une condition données, détermine l'utilité correspondante par rapport à la tâche T:

$$\begin{aligned} U_T :\ M_T \times \Theta_T &\longrightarrow \mathcal{U} \\ (m, \theta) &\longmapsto u = min_{w \in W_\theta} \bar{U}_T([m, w]) \end{aligned}$$

Propriétés de l'utilité d'une méthode

D'après ce qui précède, l'utilité d'une méthode pour une condition qui ne permet pas de l'activer est fixée à $-\infty$:

$$\forall m \in M_T\ \forall \theta \in \Theta_T\ (\theta_m <_\theta \theta) \Longrightarrow (U_T(m)(\theta) = -\infty)$$

L'utilité est une fonction décroissante par rapport à θ. Cette propriété exprime le fait que plus la condition θ est restreinte, plus l'information relative à l'état courant est complétée et plus l'utilité est importante:

$$\forall m \in M_T\ \forall \theta_1, \theta_2 \in \Theta_T\ (\theta_1 \leq \theta_2) \Longrightarrow (U_T(m)(\theta_2) \preceq U_T(m)(\theta_1))$$

La preuve de cette propriété est immédiate:

$$(\theta_1 \leq \theta_2) \Longrightarrow (W_{\theta_1} \subseteq W_{\theta_2})$$
$$(W_{\theta_1} \subseteq W_{\theta_2}) \Longrightarrow (min_{w \in W_{\theta_2}} \bar{U}_T([m, w]) \preceq min_{w \in W_{\theta_1}} \bar{U}_T([m, w]))$$

La fonction U_T est donc compatible avec la relation de spécialisation et peut être utilisée dans les inférences basées sur la classification d'instances.

Condition de sélection

On introduit la notion de condition de sélection pour spécifier les sous-ensembles d'états pour lesquels l'utilité de certaines méthodes est connue. Soit Θ_T^s l'ensemble des conditions de sélection pour la tâche T :

$$(\theta \in \Theta_T^s) \Longleftrightarrow (\exists m \in M_T \; \exists u \in \mathcal{U} \; (U_T(m)(\theta) = u) \wedge (u > -\infty))$$

L'introduction des conditions de sélection rajoute des nouveaux nœuds dans le graphe de subsumption des conditions d'activation.

Il est possible d'exprimer l'utilité de la plus petite condition supérieure à deux conditions de sélection. Si θ_1 et θ_2 sont deux conditions de sélection pour une méthode m, alors $\theta_1 \vee \theta_2$ est une condition de sélection pour m :

$$(U_T(m)(\theta_1) = u_1 \wedge U_T(m)(\theta_2) = u_2) \Longrightarrow (U_T(m)(\theta_1 \vee \theta_2) = min(u_1, u_2))$$

Cependant, le graphe défini par la relation d'ordre partiel sur l'ensemble des conditions d'activation et de sélection d'une tâche ne constitue pas un treillis dans le cas général. En effet, si l'utilité est connue pour deux sous-ensembles d'un ensemble d'états, elle peut ne pas l'être pour cet ensemble. Ainsi, il n'est pas possible de calculer l'utilité pour toutes les conditions d'activation.

Fonction de sélection

La fonction de sélection détermine pour une condition de sélection de la tâche T le sous-ensemble des méthodes dont l'utilité est maximale :

$$\begin{array}{rccl} \mathcal{S}_T : & \Theta_T^s & \longrightarrow & 2^{M_T} \\ & \theta & \longrightarrow & \{m / U_T(m)(\theta) = max_{m' \in M_T} U_T(m')(\theta)\} \end{array}$$

La fonction de sélection fournit une méthode activable :

$$\forall \theta \in \Theta_T^s \; : \; U_T(S_T(\theta))(\theta) > -\infty$$

Pour utiliser la classification, il faut définir la fonction de sélection quelle que soit la classe obtenue comme classe la plus spécialisée pour l'instance qui modélise l'état courant de la résolution. Pour obtenir ce résultat, on introduit la fonction sup_s qui à toute condition associe la plus petite condition de sélection supérieure ou égale à cette condition. Il y a bien unicité à cause de la restriction imposée pour l'absence de multispécialisation dans le graphe des conditions.

$$\forall \theta \in \Theta_T : sup_s(\theta) = min_{\leq_\theta}\{\theta_s \in \Theta_T^s / \theta_s \geq_\theta \theta\}$$

L'utilité étant décroissante par rapport à θ, si θ est une condition d'activation, alors $sup_s(\theta)$ est la condition de sélection supérieure à θ dont l'utilité est maximale. La fonction de sélection est alors étendue sur l'ensemble des conditions d'activation.

$$
\begin{array}{rrcl}
\mathcal{S}_T : & \Theta_T & \longrightarrow & 2^{M_T} \\
& \theta \in \Theta_T^s & \longrightarrow & \{m/U_T(m)(\theta) = max_{m' \in M_T} U_T(m')(\theta)\} \\
& \theta \in \Theta_T^a - \Theta_T^s & \longrightarrow & \mathcal{S}_T(sup_s(\theta))
\end{array}
$$

Algorithme de sélection

Lorsque le gestionnaire de tâches fixe la tâche courante à réaliser, une instance est créée dans le concept de sélection de cette tâche. Les attributs de cette instance sont valués par les valeurs des entrées de la tâche. Elle décrit donc l'état courant de la résolution par rapport à la tâche courante.

La sélection de la méthode dont l'utilité est maximale pour l'état courant de la résolution est alors obtenue par classification d'un objet qui modélise cet état. Les classes les plus spécialisées dont cet objet est instance fournissent au moyen de la fonction de sélection $\mathcal{S}_T$ l'ensemble des méthodes à sélectionner.

L'ensemble des classes résultat de la classification est non vide car l'objet classé est nécessairement instance de la classe racine du concept d'activation. Si les classes d'appartenance de l'instance classée ne corresondent pas à des conditions de sélection, mais seulement à des conditions d'activation, alors aucune décision n'est possible à partir de l'information disponible. On sait seulement que certaines méthodes sont activables, sans pouvoir comparer leurs utilités par rapport à la résolution. Pour éviter ce blocage, il est nécessaire d'associer à la classe racine du concept au moins une méthode dont on connait l'utilité

La méthode par défaut qui est associée à cette classe racine est donc toujours activable. Si l'ensemble des classes les plus spécialisées n'est pas réduit à une seule classe (cas de la multi-spécialisation), alors les méthodes associées à ces classes par la fonction $\mathcal{S}_T$ sont en conflit. Ce conflit doit être résolu en invoquant plusieurs critères de sélection successivement. Nous supposons dans la suite que le modèle objet ne prend pas en compte la multi-spécialisation, dans ce cas le résultat de la classification n'est constitué que de la classe la plus spécialisée.

La classe la plus spécialisée obtenue par la classification désigne par la fonction $\mathcal{S}_T$ les méthodes qui sont les plus utiles à partir de l'état courant de la résolution.

III.3.6 Sélection multi-critères

La sélection dynamique de méthodes présentée ci-dessus ne prend en compte qu'une seule mesure d'utilité pour les méthodes qui sont associées à une tâche. Or dans bien des domaines d'application, et surtout en calcul scientifique, plusieurs dimensions interviennent pour mesurer la performance des méthodes disponibles. Selon la phase courante de la résolution, le système de contrôle doit alors privilégier l'une ou l'autre de ces dimensions. En calcul scientifique, la précision et le temps d'exécution sont deux critères qui sont souvent en opposition pour le choix d'une méthode. Typiquement, pour une même opération, la précision sera privilégiée dans un calcul qui conditionne la suite des traitements alors que la rapidité sera recherchée dans les traitements itératifs. Il est donc important de prendre en compte dans la modélisation les différents critères qui peuvent guider le choix d'une méthode.

Critères de sélection

Les critères de sélection sont les éléments d'un ensemble fini de critères. Ces critères sont utilisés pendant toute la résolution, ils ne sont donc pas dépendants de la tâche pour laquelle le mécanisme de sélection recherche une méthode à activer. Des critères possibles sont :

- Le temps d'exécution exprimé sous la forme d'une mesure constante pour chaque méthode. Cette modélisation du temps d'exécution ne prend pas en compte le fait qu'un calcul puisse être paramétré pour rechercher un compromis entre la rapidité et la précision. Il est cependant possible de décomposer ce critère de sélection en plusieurs critères qui correspondent à des niveaux différents de précision.

- La précision est une mesure qui dépend de l'application. Elle peut par exemple correspondre à une valeur positive qui représente la valeur de l'erreur du calcul. En robotique, la précision peut être liée au capteur utilisé pour le calcul de la position du robot. Ce critère peut lui aussi être décomposé pour faire intervenir le temps d'exécution.

- La place mémoire requise par l'exécution du calcul constitue un critère possible dans le cas où plusieurs calculs s'exécutent en parallèle. La place mémoire disponible pour l'exécution d'une tâche donnée peut alors varier selon les autres tâches qui sont en cours d'exécution.

De facon générale, les critères interviennent pour paramétrer dynamiquement et explicitement le choix d'une méthode dans un état donné de la résolution.

Utilité d'une méthode pour un critère de contrôle

L'utilité d'une méthode dépend de critères de sélection qui sont introduits pour privilégier l'une ou l'autre des différentes dimensions du conflit des méthodes. Soit S_C l'ensemble des critères de sélection. L'utilité d'une méthode pour un critère de sélection s est à valeur dans un ensemble $(U_s, \preceq_s)$ muni d'une relation d'ordre total. On note $\bar{U}_T([m,w])(s) \in U_s$ l'utilité d'exécuter la méthode m pour réaliser la tâche T à partir de l'état w et pour le critère de sélection s. Pour le sous-ensemble des états spécifiés par la condition θ, l'utilité d'une méthode s'exprime donc par :

$$U_T(m)(\theta)(s) = min_{w \in W_\theta} \bar{U}_T([m, w])(s)$$

Il n'est pas possible de comparer les utilités d'une méthode pour des critères de sélection différents. Cependant, le critère de sélection étant fixé, l'utilité d'une méthode est une fonction décroissante par rapport à θ :

$$\forall m \in M_T \ \forall \theta_1, \theta_2 \in \Theta_T \ (\theta_1 \leq \theta_2) \Longrightarrow (U_T(m)(\theta_2)(s) \preceq_s U_T(m)(\theta_1)(s))$$

Fonction de sélection multi-critères

La fonction de sélection est alors paramétrée par les critères de sélection :

$$\begin{array}{rccl} \mathcal{S}_T : & \Theta_T \times S_C & \longrightarrow & 2^{M_T} \\ & (\theta, s) & \longmapsto & \{m \in M_T / U_T(m)(\theta) = max_{m' \in M_T} U_T(m')(\theta)(s)\} \end{array}$$

De même que pour un seul critère, la fonction de sélection multi-critères fournit une méthode activable.

Critères de contrôle et points de vue

Dans ce qui précède, nous avions supposé que le graphe des conditions (d'activation et de sélection) des méthodes associées à une tâche était une hiérarchie. En fait, cette restriction n'est acceptable que pour le sous-ensemble des méthodes associées à une tâche qui sont les méthodes les plus utiles par rapport à un seul critère de contrôle. Si l'on considère l'ensemble de toutes les méthodes associées à une tâche, alors le graphe des conditions n'est pas une hiérarchie.

Le modèle objet que nous avons proposé dans le chapitre II n'accepte pas la multi-spécialisation. Nous avions justifié cette restriction en faisant référence aux points de vue qui permettent de décomposer le graphe de spécialisation des classes d'un concept en plusieurs sous-graphes qui sont sans multi-spécialisation. A chaque critère de contrôle, il est donc possible d'associer un point de vue dont le graphe de spécialisation constitue la hiérarchie de sélection pour ce critère. L'algorithme de classification multi-points de vue utilise des passerelles pour passer d'un point de vue à l'autre [Mari93]. Le modèle objet TROPES met en œuvre ce type de classification [Mari90].

Algorithme de sélection multi-critères

La sélection se fait à partir d'un ordre total sur les critères de sélection : $s_1, ..., s_n$ étant rangés par ordre de priorité décroissante. Le résultat de la classification est donné par la classe la plus spécialisée à laquelle appartient l'instance qui modélise l'état de la résolution. Par applications successives de la fonction $\mathcal{S}_T$ à cette classe, de l'ensemble des méthodes associées à la classe c sont retenues celles qui sont le plus utiles selon le critères s_1, puis parmi celles-ci sont retenues celles qui sont les plus utiles selon s_2, et ainsi de suite jusqu'à l'utilisation de tous les critères de sélection ou jusqu'à l'obtention d'un ensemble réduit à une seule méthode. Celle-ci a l'utilité maximale par rapport à l'ordre défini sur les critères de sélection. Ainsi, la recherche de la précision dans les calculs favorisera d'abord les méthodes les plus précises, et en cas de conflit seront retenues les méthodes les plus rapides.

Si le modèle objet propose la classification multi-points de vue, alors l'ordre sur les critères de contrôle ordonne les points de vue définis par les critères, comme c'est le cas dans le système TROPES [Mari90].

III.3.7 Evaluation de cette approche

L'évaluation de cette approche concerne plusieurs éléments.

Calcul de l'utilité des méthodes

La difficulté de ce genre d'approche est due aux valeurs numériques qui interviennent dans l'expression de l'utilité. Les temps d'exécution peuvent être calculés dans le cas de modules de calcul scientifique. Les probabilités peuvent être évaluées au cours d'une phase d'apprentissage [Etzi91]. Il est plus difficile d'attribuer une valeur numérique à la précision, or l'utilité doit être une valeur additive.

Acquisition des connaissances

L'intérêt de cette approche apparaît lors de la phase d'acquisition des connaissances. En effet, la hiérarchie de spécialisation des conditions d'activation des méthodes est un outil de description puissant et facile à utiliser. Il ne nécessite pas de faire référence au système d'objets mis effectivement en œuvre pour la résolution. Selon les niveaux de Newell [Newe82], la spécification des méthodes d'une tâche est faite au niveau des connaissances.

Sélection par classification et réactivité

Les séquenceurs de tâches réalisés à partir de cette approche de la sélection dynamique de méthodes ne constituent pas des systèmes réactifs.

- La classification d'instance pour sélectionner une méthode parmi celles qui réalisent la tâche courante recherche systématiquement celle qui est optimale. Cette recherche peut s'avérer coûteuse en ressources (temps et place mémoire), ce qui est incompatible avec la réactivité. Il est donc intéressant d'adapter cette approche aux exigences d'un système réactif.

– Cette approche ne concerne que les méthodes de la tâche dont l'exécution a été explicitement demandée par le module de contrôle, et ne détecte pas les possibilités d'activation des méthodes qui ne sont pas associées à cette tâche. Cette approche de la sélection dynamique de méthodes ne permet donc pas de gérer les possibilités d'activation.

Nous proposons donc en section III.4 un mécanisme de classification rationnelle d'instances afin d'adapter cette approche aux contraintes de temps, et en section III.5 une autre approche qui détecte les possibilités d'activation de toutes les méthodes disponibles.

III.4 Sélection rationnelle de méthodes

Cette section présente une adaptation de l'approche précédente aux besoins d'un système réactif [Chai93]. Cette nouvelle approche est basée sur l'utilisation de la théorie de la décision pour contrôler le déroulement de la classification afin de mettre en œuvre un mécanisme de sélection rationnelle de méthodes.

III.4.1 Délibérations rationnelles

La mise en œuvre de mécanismes d'inférence dans un système temps réel pose le problème de la maîtrise de ces mécanismes. Parmi les solutions utilisées pour concevoir des systèmes à base de connaissances temps réel (voir I.3.4), la rationalité limitée (ou rationalité économique) se préoccupe des limitations des ressoures du système lors de la prise de décisions. Une décision est prise à partir des possibilités qui sont déterminées par un mécanisme d'inférence. Cette inférence consomme des ressources, en particulier le temps utilisé pour les calculs. Les heuristiques sont utilisées pour optimiser la gestion de ces ressources. La rationalité [Doyl89] se préoccupe de formaliser et de prouver les heuristiques en utilisant la théorie de la décision.

Parmi les applications de l'approche rationnelle pour contrôler les mécanismes d'inférence, les résultats les plus avancés concernent le contrôle d'une délibération par rapport au temps consommé. S. Russell et E. Wefald proposent une modélisation du contrôle rationnel d'une délibération ("méta-raisonnement" [Russ91]) qu'ils appliquent aux problèmes du temps réel. T. Dean reprend cette modélisation par rapport à plusieurs buts et toujours dans le contexte du temps réel [Dean91].

Décision de méta-niveau Le problème de la décision de méta-niveau consiste à choisir entre poursuivre les calculs dans le but d'améliorer l'analyse de la situation ou se contenter des résultats obtenus pour agir. La poursuite des calculs a pour objectif d'agir de façon optimale par rapport à la situation (principe de rationalité logique) et donc de sélectionner une action plus performante que celle qui est actuellement sélectionnée. L'évaluation d'une action par rapport à l'état de la résolution est faite par son utilité. Une action du système constitue sa réponse à un événement et l'utilité de cette réponse décroît avec le temps qui s'écoule. Donc prolonger les calculs pour améliorer l'utilité intrinsèque de la réponse diminue son utilité relativement à la ressource temps qui est consommée par les calculs. Il est donc nécessaire de rechercher un compromis pour ne poursuivre les calculs que s'ils sont prometteurs : leurs probabilités de succès sont suffisement grandes pour envisager un gain.

Notations Les calculs constituent également des actions du système qui n'affectent que son état interne. S. Russell et E. Wefald distinguent donc les actions calculatoires et les actions physiques. A chaque étape de la délibération, le système est confronté avec les actions calculatoires S_i et l'action physique α. Soit $U([A_i, W_k])$ l'utilité d'exécuter une action (calculatoire ou physique) A_i à partir d'un état W_k. La théorie de la décision identifie l'action optimale comme étant celle qui maximise son utilité espérée. L'utilité espérée d'une possibilité est la somme pondérée de ses utilités dans les différents états possibles :

$$E[U([A_i])] = \sum_k P(W_k) U([A_i, W_k])$$

où $P(W_k)$ est la probabilité que le système soit réellement dans l'état W_k.

Evaluation de l'utilité L'utilité d'une action physique est une grandeur directement rattachée à l'application. L'utilité d'une action calculatoire dépend des changements qui surviennent pendant ce calcul et de ses effets sur l'état interne du système, effets qui peuvent le conduire à modifier la sélection de l'action physique. L'utilité d'un calcul S_i est donc l'utilité pour le système d'atteindre l'état $[S_i]$ qui résulte de ce calcul. La valeur nette d'un calcul est le gain en utilité qui est obtenu par ce calcul par rapport à l'action physique α : $V(S_i) = U([S_i]) - U([\alpha])$

Deux types de calculs sont à prendre en compte : les calculs *complets* qui aboutissent directement à la sélection d'une nouvelle action α_{S_i} plus utile que l'action α et les calculs *partiels* qui ne constituent qu'une étape d'un algorithme et ne modifient donc pas la sélection de l'action α.

- L'utilité d'un calcul complet S_i est l'utilité d'exécuter l'action α_{S_i} à partir de la situation $[S_i]$ qui résulte de ce calcul (avec en particulier du temps qui s'est écoulé) : $U([S_i]) = U([\alpha_{S_i}, [S_i]])$. Donc la valeur nette d'un calcul complet S_i est :

$$V(S_i) = U([\alpha_{S_i}, [S_i]]) - U([\alpha])$$

- Un calcul partiel ne modifie que l'état interne du système; son utilité ne peut donc être directement calculée, car elle dépend de son effet par rapport aux calculs ultérieurs. Les calculs correspondent au parcours d'un graphe avec des tests pour progresser dans le graphe. Ces tests peuvent échouer, ce qui correspond alors à une perte de temps. Il n'est donc possible de calculer que son utilité espérée en fonction des différents calculs ultérieurs et de leurs conclusions. Ces calculs s'effectuent en séquence et une séquence de calculs est complète si son dernier calcul est complet, donc si elle modifie la sélection de la méthode α. Soit $\mathcal{T}$ l'ensemble des séquences complètes de calculs à partir de l'état $[S_i]$ et soit T_α l'action sélectionnée par une séquence complète T. L'utilité de l'état $[S_i]$ est alors :

$$[S_i] = \sum_{T \in \mathcal{T}} P(T)U([\alpha_T, [S_i.T]])$$

où $[S_i.T]$ est l'état qui résulte de la séquence $S_i.T$.

Contrôle rationnel idéal L'algorithme de contrôle rationnel obtenu par ces résultats est composé de deux étapes :

1. Exécuter le calcul dont l'utilité espérée est la plus grande tant qu'il y a des calculs dont l'utilité espérée est positive.

2. Exécuter l'action physique α qui résulte des calculs précédents.

Le contrôle ainsi défini est *idéal* dans la mesure où les grandeurs numériques utilisées sont des approximations de leurs valeurs réelles. S. Russell et T. Dean reformulent l'expression de l'utilité des calculs en utilisant des variables aléatoires pour les grandeurs numériques qui interviennent dans l'utilité espérée.

O. Etzioni [Etzi89] utilise ces résultats dans le cas d'un ensemble de méthodes m_i qui peuvent être testées pour tenter de réaliser plusieurs objectifs simultanés. Il propose par ailleurs [Etzi91] une approche basée sur la classification numérique pour construire les classes d'états connues et pour calculer leurs probabilités d'appartenance. E. Horvitz [Horv89] définit un réseau compilé pour mettre en œuvre le contrôle rationnel et pour apprendre les probabilités utilisées. Ces travaux montrent que l'utilisation de la théorie de la décision permet de démontrer l'optimalité des délibérations et que cette approche, associée aux techniques de compilation, permet d'aborder les problèmes du temps réel pour les systèmes à base de connaissances.

III.4.2 Classification et rationalité

Utiliser la classification d'instances dans une architecture de contrôle réactif et rationnelle nécessite de maîtriser ce mécanisme. La sélection de méthodes dans un système réactif privilégie les méthodes les plus utiles. La classification d'une instance dans une hiérarchie de classes consiste à déterminer la ou les sous-classes les plus spécialisées, auxquelles cette instance peut être rattachée. Le test d'appartenance d'une instance à une classe donnée constitue donc le pas de progression du mécanisme de classification. Ce test est défini par la condition d'appartenance de la classe. Cette condition porte sur les valeurs des attributs de l'instance, elle peut ainsi faire intervenir d'autres instances, si l'instance étudiée est un objet complexe. Dès lors, le coût de la classification n'est pas négligeable devant les différences entre les temps d'exécution des différentes méthodes parmi lesquelles le choix doit être effectué. Il est donc important de ne poursuivre la classification que si elle apporte un gain net positif.

Dans le cadre de la classification, le mécanisme de sélection analyse l'état courant de la résolution pour déterminer la méthode la plus intéressante pour la résolution. L'état courant est modélisé sous la forme d'une instance. L'analyse se fait en testant son appartenance aux différentes classes qui sont définies par les conditions de sélection des méthodes disponibles pour atteindre le but courant. La classe racine correspond à la condition de sélection la plus générale, donc préconise la méthode par défaut. A chaque étape de la classification, le système peut, soit interrompre la classification et se contenter d'activer la méthode qui correspond à la classe courante, soit prolonger la classification pour rechercher une méthode plus intéressante. Il a donc le choix entre une action physique α (le déclenchement de la méthode courante) et des actions de calcul, S_j. Celles-ci consistent à tester l'appartenance de l'instance aux différentes sous-classes de la classe courante. Ces tests sont de coûts différents, et de probabilités de succès différents. La classification est rationnelle, si, à chaque étape, elle sélectionne la possibilité la mieux adaptée à l'état courant de la résolution [Doyl90].

III.4.3 Sélection à deux critères: activabilité et rapidité

Dans cette section, le temps d'exécution de chaque méthode et le temps nécessaire pour tester l'appartenance d'une instance à une classe sont supposés connus et constants.

Description de la décision

La première étape de la classification teste l'appartenance de l'instance à la classe racine de la hiérarchie de spécialisation qui décrit le concept d'activation pour la réalisation de la tâche courante. Ce test est incontournable, et s'il échoue, cette tâche n'est pas réalisable. A chaque étape suivante, la classe C_α est une classe sûre pour l'instance classée, et ses sous-classes C_i sont des classes possibles. La décision est donc définie par les alternatives suivantes:

Le système peut, soit déclencher la méthode m_α dont le temps d'exécution est τ_α, soit tester l'appartenance de l'instance à l'une des sous-classes C_i de C_α. Le test S_i a la probabilité p_i de réussir, permettant alors de déclencher la

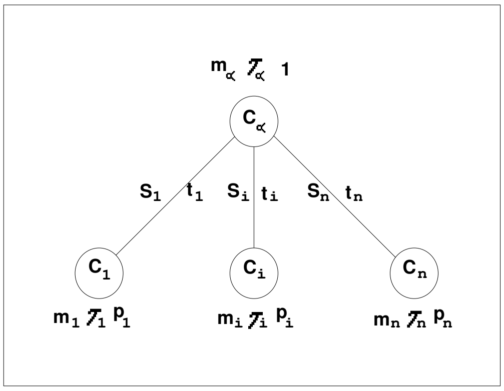

FIG. III.3 - *Description de la décision à chaque pas de la classification.*

méthode m_i dont le temps d'exécution est τ_i. Le système a donc le choix entre une action physique m_α et une action calculatoire S_i de test sans garantie de succès.

Ordre sur les tests d'appartenance aux sous-classes

Le système peut évaluer l'utilité de chacune des possibilités. Comme il ne connaît pas avec certitude la situation dans laquelle il se trouve, il utilise l'utilité espérée. Le test S_i d'appartenance de l'instance à la sous-classe C_i a une utilité espérée donnée par :

$$E(U([S_i])) = \sum_k p_k U([S_i, W_k])$$

Les situations possibles (indexées par k) correspondent à l'appartenance de l'instance aux différentes sous-classes de C_α, et au cas particulier où elle n'appartient à aucune d'entre elle (cas où la classe C_α n'est pas égale à l'union de ses sous-classes). Le test S_i n'est satisfait que dans le cas où l'instance peut être rattachée à la classe correspondante (notée C_i). En cas d'échec, son utilité est la perte de temps t_i utilisé pour effectuer le test.

$$E(U([S_i])) = p_i((\tau_\alpha - \tau_i) - t_i) - t_i \sum_{k \neq i} p_k = p_i \times \Delta_i - t_i \sum_k p_k$$

$\Delta_i = (\tau_\alpha - \tau_i)$ désigne le gain en temps de la méthode m_i par rapport à la méthode m_α. La somme des probabilités est égale à 1 puisque toutes les situations possibles sont prises en compte.

L'utilité espérée maximale correspond au test optimal S_i^* tel que :

$$E(U([S_i^*])) = max_k(p_k \times \Delta_k - t_k)$$

En ordonnant les tests par leur utilité espérée, on fixe le test qui sera effectué en premier. Quand le premier test a échoué, une nouvelle décision est à prendre :

les tests ont alors une utilité espérée donnée par :

$$E(U([S_i])) = p_i((\tau_\alpha - \tau_i) - (t_i - t_1^*)) - t_i \sum_{k \geq 2} p_k$$

où p_i est la probabilité que l'instance appartienne à la sous-classe C_i sachant qu'elle n'appartient pas à la sous-classe C_1^*. La somme de ces probabilités est toujours égale à 1. Après l'échec des q premiers tests, le critère d'ordonnancement des tests restants devient :

$$E^q(U([S_i])) = p_i^q \times \Delta_i^q - t_i$$

où p_i^q désigne la probabilité que le test S_i soit satisfait sachant que les tests S_j^* $(j \leq q)$ ont échoué, et $\Delta_i^q = \Delta_i - \sum_{j=1}^{q} t_j^*$ introduit dans le gain la perte de temps due aux échecs précédents.

Arrêt de la classification et regret

La classification est interrompue soit lorsque l'instance ne peut plus être classée (tous les tests ont échoué, ou bien C_α est une feuille), soit lorsqu'elle n'est plus intéressante. Ce dernier cas intervient quand l'utilité espérée de tous les tests restants est négative :

$$max_{i \leq q}[p_i(\Delta_i^q - t_i)] \leq 0$$

Le regret δ de cette interruption de la classification est l'utilité maximale qui aurait pu être gagnée en continuant les tests :

$$\delta = max_{i \leq q}(\Delta_i^q - t_i)$$

Ce regret est calculé en ne prenant en compte qu'un seul pas supplémentaire dans la classification.

III.4.4 Evaluation de cette approche

La difficulté de ce type d'heuristique est liée à l'obtention des valeurs numériques utilisées pour ordonner les différentes alternatives. Le temps d'exécution des méthodes est connu a priori s'il s'agit de codes de calcul scientifique. Les probabilités de succès des tests d'appartenance d'une instance aux différentes classes peuvent être calculées dans une phase d'échantillonage. La précision d'une méthode est plus difficile à quantifier, le gain associé aux différentes méthodes devant être une grandeur additive.

La plupart des travaux [Dean91] [Horv89] [Ingr92] [Russ91] consacrés à la rationalité d'un agent s'attachent à construire une séquence de méthodes pour réaliser un ensemble de buts. Ces buts ne sont pas conjonctifs comme dans la problématique de la sélection de méthodes dans un $\mathcal{ERP}$, mais à valeur additive : l'échec d'un but ne condamne pas les autres, et les gains associés aux différents succès s'ajoutent. Les méthodes sont essayées les unes après les autres jusqu'à l'obtention d'un succès, ainsi la délibération n'exploite pas les implications entre leurs conditions d'activation. Ces travaux ne prennent pas en compte les relations d'ordre partiel qui peuvent exister entre les différentes méthodes d'un même but. Etzioni [Etzi89] fait référence à la relation de dominance : une méthode m_i domine une méthode m_j si dans toutes les dimensions (rapidité, précision ...) elles ont les mêmes caractéristiques, et si, pour une dimension, m_i est meilleure que m_j. Cette relation de dominance ne correspond pas aux modules de calcul scientifique, dont la rapidité et la précision varient inversement et obligent à rechercher un compromis.

III.5 Système de production d'objets

Nous présentons dans cette section le principe de fonctionnement d'un système de production d'objets. Comme dans tous les systèmes de production, un mécanisme d'évaluation caractérise l'état courant de la base de données. Dans le cas d'un système de production d'objets, cette évaluation combine la classification d'instances et la détection d'événements (Figure III.4).

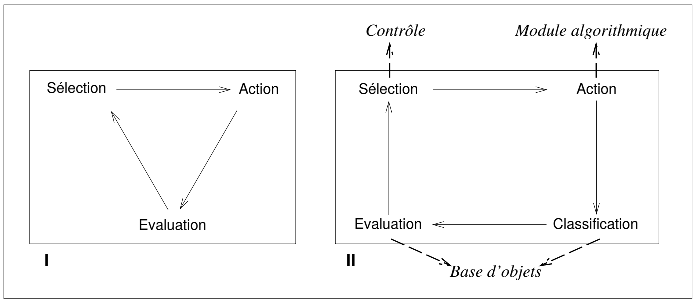

FIG. III.4 - *Comparaison des systèmes de production à base de règles (I d'après [Fage92]) et des systèmes de production d'objets (II).*

Nous montrons comment ce type de système permet de définir un contrôle réactif, de gérer les possibilités d'activation et de mettre en œuvre le contrôle opportuniste qui consiste à exploiter les possibilités d'activation.

III.5.1 Configurations et réactivité

Les configurations définies dans notre modèle objet (II.5) permettent de caractériser l'état courant de la base qui contient la modélisation des entités de la résolution. Cette caractérisation consiste à détecter les événements qui correspondent à la validité des configurations. Ces événements sont utilisables pour définir un contrôle réactif en leur associant des transitions dans l'automate d'états finis qui met en œuvre le comportement réactif du système.

Un événement défini par la validité d'une configuration correspond à l'identificateur de cette configuration et aux données qui la valident. La transition peut être définie sur l'identificateur mais pas directement sur les données. Il faut tester les données pour que la transition soit effectivement validée.

Les événements sont donc définis par les configurations et stockés sous la forme des tuples d'instances qui valident ces configurations.

III.5.2 Configuration d'activation d'une action

Pour toutes les actions pilotées par le système, on définit une condition d'activation de la même manière que pour les méthodes associées aux tâches. En fait, cette section ne fait référence qu'à des actions car les tâches ne sont pas prises en compte, mais il s'agit bien des mêmes modules algorithmiques.

Pour détecter les événements qui correspondent à des possibilités d'activation, il faut associer des configurations aux conditions d'activation des actions.

Rappel Le contexte d'activation d'une méthode (III.3.2) est un vecteur de variables qui interviennent pour spécifier le comportement de la méthode. Ces variables font référence à des propriétés d'entités de la résolution. Lorsque les données de la résolution sont représentées dans un modèle objet, les données qui correspondent à ces variables sont contenues dans les instances qui modélisent les entités concernées.

Dès lors, le contexte d'activation d'une action permet de définir le contexte d'une nouvelle configuration de la même manière qu'il permet de définir le concept d'activation d'une tâche (III.3.3).

En associant à la condition d'activation de chaque action une configuration, tous les tuples d'intances qui permettent d'activer une action deviennent des éléments de la configuration associée à cette action. Les configurations qui sont associées à l'activabilité d'une action sont appelées des configurations d'activation.

III.5.3 Détection des possibilités d'activation

Le principe de fonctionnement d'un système de production d'objets est de caractériser tous les objets de la base relativement aux événements qui doivent être détectés. Tous les objets créés ou modifiés sont classés dans la taxonomie de spécialisation de leur concept. Cette classification a pour effet de faire migrer ces objets, qui ainsi valident de nouvelles configurations et entre autres des configurations d'activation (Figure III.5).

III.5.4 Gestion des possibilités d'activation

La gestion des possibilités d'activation concerne le stockage et l'accès aux possibilités d'activation. Telles que les configurations sont définies dans le chapitre II, les données qui valident une configuration d'activation sont contenues dans le tuple d'instances qui correspond à cette possibilité d'activation.

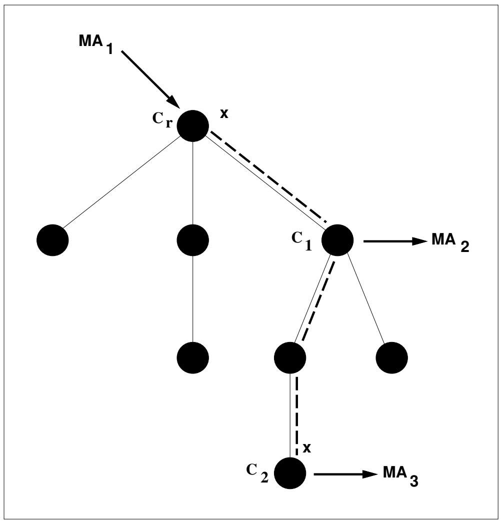

FIG. III.5 - *Principe de fonctionnement d'un système de production d'objets pour la détection des événements qui correspondent à l'activabilité de modules algorithmiques: un objet x est créé par une action MA_1, il est rattaché à la classe racine c_r de son concept; lors de la classification, l'objet est successivement rattaché aux classes c_1 et c_2, ce qui rend activable respectivement les actions MA_1 et MA_2.*

Agenda Le stockage des possibilités d'activation est donc assuré par la classe qui est associée à chaque configuration d'activation. Les instances de ces classes sont les tuples d'instances qui valident la configuration. Les données qui rendent activable une action sont les valeurs des attributs des instances du tuple correspondant à cette possibilité d'activation. Ainsi, l'ensemble des instances d'une classe définie par une configuration d'activation constitue l'agenda des possibilités d'activation de l'action correspondante.

Accès aux possibilités d'activation L'accès aux possibilités d'activation d'une action se fait à partir de son agenda. La structure d'agenda qui contient les possibilités d'activation de l'action à déclencher doit donc être parcourue pour retrouver celle qui correspond à la phase courante de la résolution et en extraire les valeurs des paramètres d'appel. Cette recherche dans la structure d'agenda est définie par les critères de contrôle qui spécifient les paramètres effectifs de l'appel. Nous avons défini deux critères de contrôle pour identifier, parmi les possibilités d'activation d'une action, celle qui correspond aux paramètres effectifs :

1. Le critère de flot de données consiste à spécifier les paramètres par rapport au flot des entrées-sorties des actions qui ont déjà été exécutées.

2. Le critère de relation d'ordre définit une relation d'ordre total sur les possibilités d'activation de l'action. Il est immédiat que les critères d'ancienneté et de valeurs d'un paramètre sont intéressants pour définir un

mode de recherche des données dans l'agenda.

III.5.5 Contrôle opportuniste

Nous définissons l'opportunisme comme un cas particulier de la réactivité : le système réagit à des événements particuliers qui sont les possibilités d'activation.

Dans l'organisation du contrôle réactif à partir d'un automate d'états finis, l'opportunisme est obtenu en définissant des transitions qui deviennent valides pour des événements qui correspondent à l'activabilité des actions concernées. Une telle transition ne peut être spécifiée par rapport aux données qui rendent activable l'action, mais seulement par rapport à l'activabilité. Ainsi, un événement d'activation ne contient comme information que l'activabilité de l'action et non les données concernées. Si l'action n'admet pas de paramètres d'appel, alors l'activabilité suffit à déclencher l'action. Si elle admet des paramètres d'appel, pour activer l'action il faut donc accéder à ces données pour en extraire les paramètres d'appel.

Le contrôle opportuniste est composé de cinq étapes :

1. Le module de gestion des possibilités d'activation détecte un ensemble de données qui rend activable une action.

2. Cet événement est transmis à la couche réactive et valide la transition qui lui est associée.

3. Si la priorité des transitions le spécifie, la transition est tirée et un nouvel état de l'automate devient valide.

4. Cet état spécifie d'une part l'action qui doit être activée, et d'autre part le critère de contrôle utilisé pour rechercher les éventuels paramètres d'appel de cette action.

5. Les éventuels paramètres de l'action sont recherchés dans l'agenda en utilisant le critère de contrôle et s'ils sont obtenus, alors l'action est déclenchée.

III.5.6 Evaluation de cette approche

Cette description de la mise en œuvre du mode de contrôle opportuniste met en évidence son coût : la recherche de la possibilité d'activation dans l'agenda est pénalisante, et si des données qui rendent activable une action sont régulièrement créées dans la base, alors une succession d'événements d'activabilité peut apparaître. Si ces données ne sont pas celles qui doivent être utilisées pour déclencher l'action, cette recherche sera répétée inutilement.

Cette approche est donc essentiellement utilisable pour les méthodes qui s'exécutent sans paramètres vis-à-vis du contrôle. Dès lors, c'est l'événement qui correspond seulement à la possibilité d'activation qui est pris en compte dans le contrôle. La méthode est exécutée sans que le contrôle ne vérifie le contenu de ses tuples d'activation.

III.6 Approche mixte

La confrontation des deux types d'approches décrites ci-dessus montre qu'elles sont complémentaires :

- La sélection par classification d'instances (rationnelle ou non) est bien adaptée au contrôle à partir de plans d'actions. La sous-tâche courante détermine un ensemble de méthodes et la sélection est faite en fonction des valeurs des paramètres d'appel de la tâche.

- Les systèmes de production d'objets caractérisent les données de la résolution pour identifier les actions qu'elles rendent activables. Ce fonctionnement permet de détecter et de gérer les possibilités d'activation des actions.

Si le contrôle opportuniste est intéressant (voir I.2.7) en résolution de problèmes, il est insuffisant pour aborder les applications telles que le pilotage de modules algorithmiques en informatique scientifique. En effet, de nombreux plans d'actions existent et doivent être appliqués pour guider la résolution, ce qui est incompatible avec un contrôle exclusivement opportuniste. Par contre, la détection et la gestion des possibilités d'activation est intéressante : elle permet d'évaluer le rôle de chaque action par rapport aux plans d'actions qui sont appliqués pour éventuellement modifier ces plans d'actions.

Il est donc souhaitable de combiner les deux approches pour mettre en place un mécanisme de sélection dynamique de méthodes qui gère les possibilités d'activation tout en étant compatible avec l'approche de sélection hiérarchique.

III.6.1 Cohabitation de deux types de méthodes

Une première approche pour associer les deux types de mécanismes de sélection dynamique de méthodes consiste à définir deux types d'actions : celles qui interviennent dans les plans d'actions et qui sont gérées par classification de l'instance qui modélise la tâche courante, et celles qui ne sont déclenchées que sur événements d'activabilité et qui sont gérées par les configurations d'activabilité.

Cette approche est simple à mettre en œuvre, mais elle ne permet pas d'atteindre l'un des objectifs recherché : l'adaptation de plans d'actions prédéfinis. En effet, pour atteindre cet objectif, il est nécessaire qu'une action, dont le comportement est d'abord caractérisé par ses événements d'activabilité, soit ensuite utilisée comme méthode d'une tâche dans un plan d'actions.

III.6.2 Intégration des deux approches

L'intégration des deux approches consiste à adapter le mécanisme de sélection hiérarchique pour gérer les possibilités d'activation.

La bibliothèque de plans et de modules algorithmiques est organisée en fonction des tâches et des méthodes de la même manière que dans la section III.3. Aux méthodes associées à une tâche correspondent les conditions d'activation et les conditions de sélection qui forment un arbre. Les variables utilisées dans cet arbre constituent les attributs du concept d'activation de la tâche et les nœuds de l'arbre définissent les conditions d'appartenance des classes qui décrivent ce concept. Ces classes sont en fait des configurations d'activation, car leurs instances sont les tuples d'instances qui valident la condition d'activation ou de sélection.

Le mécanisme de sélection dynamique de méthodes est alors le même que celui de la sélection par classification, sauf que le test d'appartenance à une classe de l'instance qui modélise l'exécution de la tâche est remplacé par la recherche dans l'agenda du tuple qui modélise l'activabilité de la tâche. Cette recherche est spécifiée par les contraintes du flot des données dans le plan d'actions dont la tâche courante est une sous-tâche. La figure III.6 décrit le rôle de la base d'objets dans cette approche.

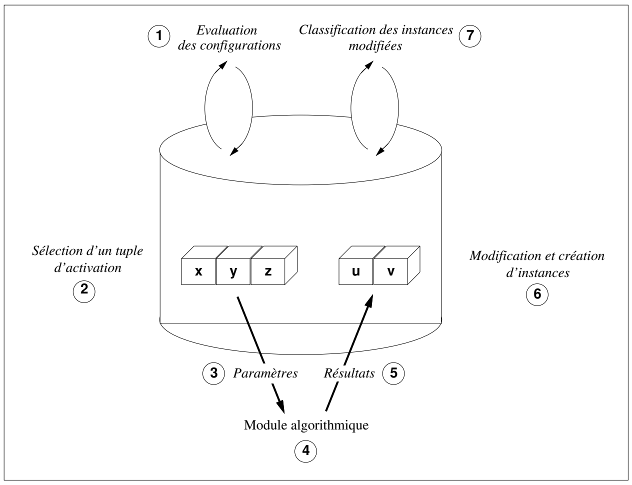

FIG. III.6 - *Rôle de la base d'objets : (1) le processus d'évaluation des configurations d'activation détermine les tuples d'instances qui rendent activables les méthodes, (2) le tuple d'instances qui vérifie les critères de contrôle est sélectionné, (3) les paramètres d'appel sont extraits des instances, (4) le module algorithmique s'exécute, (5) les résultats obtenus sont transmis à la base, (6) la mise à jour de la base modifie et créé des instances, (7) le processus de classification des instances modifiées et des nouvelles instances détermine leurs classes d'appartenance.*

III.6.3 Evaluation de cette approche

La gestion des possibilités d'activation par les configurations d'activation est intéressante, elle permet de mettre de rendre compte du rôle de chaque action pendant la résolution : les actions activables inutilisées sont détectées ainsi que celles qui sont sollicitées sans être activables. L'objectif est alors d'évaluer le rôle de chaque action par rapport aux plans d'actions appliqués pour éventuellement les améliorer.

III.7 Conclusion

Nous avons étudié dans ce chapitre plusieurs approches pour aborder le problème de la sélection dynamique de méthodes dans un $\mathcal{ERP}$ réactif. Les solutions proposées utilisent la classification d'instances comme mécanisme d'inférence pour caractériser les données de la résolution qui concernent les méthodes en conflit et la détection d'événements pour gérer les possibilités d'activation. L'importance du rôle de la classification souligne l'intérêt des systèmes de production couplés avec la classification. En utilisant la théorie de la décision pour contrôler le temps consommé par le processus de classification, nous avons défini un mécanisme de sélection rationnelle de méthodes, donc compatible avec les exigences du temps réel dans un système à base de connaissances.

La conception d'un $\mathcal{ERP}$ générique, c'est-à-dire indépendant du domaine, présuppose qu'il existe une approche générale pour modéliser les connaissances relatives aux comportements des modules d'une bibliothèque de calcul. Cette supposition n'est pas vérifiable, aussi l'organisation des tâches et des méthodes pour la sélection hiérarchique ne constitue qu'une solution possible au problème de la modélisation. La difficulté de la mise en application de notre approche est principalement liée à l'expression de l'utilité. Par ailleurs, il serait intéressant de valider cette approche pour la modélisation des codes de calcul monolithiques : ils intègrent plusieurs sous-modules dont ils contrôlent l'exécution par l'intermédiaire de leurs nombreux paramètres.

Chapitre IV

Réalisation et validation

Si la modélisation des connaissances de résolution conditionne la capacité d'un $\mathcal{ERP}$ à aborder des problèmes non triviaux, les aspects du génie logiciel relatifs à l'intégration des bibliothèques de calcul dans un tel système conditionnent l'aboutissement des applications en vraie grandeur. La réutilisabilité est donc nécessaire pour évaluer l'approche proposée. Or les problèmes liés à l'intégration de modules logiciels externes dans un système à base de connaissances ne sont pas simples et interviennent au niveau de l'architecture logicielle du système obtenu.

Ce chapitre présente la réalisation de TRAM [Chai92a], et les applications qui permettent de l'expérimenter. Nous présentons tout d'abord l'organisation du contrôle réactif et de planification (IV.1). L'architecture client-serveur qui la met en œuvre est alors détaillée (IV.2). Nous décrivons les applications qui permettent de valider cette architecture (IV.3). Enfin, nous proposons une perspective pour l'explication dans un $\mathcal{ERP}$ réactif (IV.4).

IV.1 Définition du contrôle

Le contrôle dans TRAM est organisé en deux modules qui gèrent respectivement la planification et la réactivité. Cette section décrit les aspects liés à la réalisation de ce type de contrôle.

IV.1.1 Superviseur

L'organisation de la couche réactive du contrôleur de TRAM est basée sur la discrétisation des situations possibles du système (voir I.2) et sur les transitions entre ces différentes classes de situations. La gestion de ces transitions est adaptée aux caractéristiques d'un $\mathcal{ERP}$: les événements qui leur sont associés peuvent concerner l'exécution des tâches, des plans d'actions ou des modules algorithmiques. La gestion de l'automate est à la charge du superviseur.

Classes de situations et états Soit $\{s_1, ..., s_n\}$ l'ensemble fini des classes de situations s_i qui décrivent chacune un sous-ensemble de situations possibles de la résolution. Les classes de situations ne sont définies ni en extension, ni en intention, mais seulement identifiées comme référents de leurs ensembles de situations.

Afin de prendre en compte toutes les situations possibles de la résolution, s_{Top} désigne la classe qui recouvre toutes les situations possibles de la résolution. L'ensemble fini des classes de situations est muni d'une relation d'ordre partiel $\sqsubseteq$ définie par la relation d'inclusion.

$s_i \sqsubseteq s_j$ si et seulement si la classe de situations s_i est incluse dans la classe de situations s_j. Toute classe de situations s_i est donc inférieure à s_{Top}.

$$\forall s_i \in \{s_1, ..., s_n\}\ s_i \sqsubseteq s_{Top}$$

L'ensemble S des classes de situations est donc organisé sous la forme d'un arbre dont la racine est la classe s_{Top}.

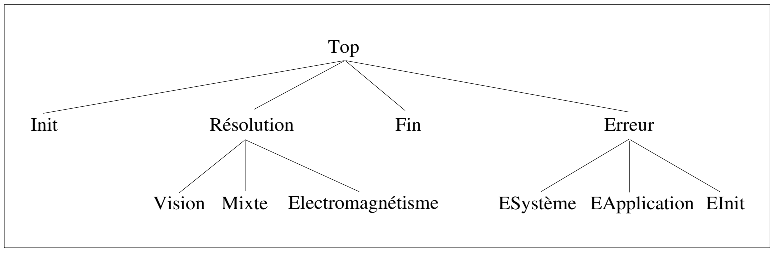

FIG. IV.1 - *Organisation en arbre des classes de situations qui discrétisent l'ensemble des situations possibles de la résolution.*

La définition du contrôle réactif est faite à partir des nœuds de l'arbre des classes de situations, mais les états de l'automate qui gère la réactivité sont définis à partir des feuilles de cet arbre. L'ensemble des feuilles de l'arbre des classes de situations est noté $\mathcal{S}_e$, c'est lui qui contient l'ensemble des états de l'automate d'états finis.

Evénements et configurations Les transitions sont introduites afin de spécifier le comportement réactif du système pendant la résolution. Elles sont définies à partir de la détection d'événements et leur associent les changements d'états qui correspondent au comportement correct du système par rapport à ces événements.

Soit $\mathcal{E}$ l'ensemble des événements observés. Ces événements sont de différentes natures :

- Evénements issus de l'environnement extérieur : ils sont détectés à partir des données fournies par les capteurs. En surveillant ces données, le système peut éventuellement modifier la résolution pour leur apporter une réponse.
- Evénements de résolution : ils concernent la progression de la résolution. Ils peuvent correspondre à l'obtention d'un résultat particulier ou à l'apparition d'une nouvelle possibilité d'action.
- Evénements système : ils sont internes au système de contrôle, ils surviennent lors du dépassement d'une limite en temps, en cycle de contrôle ou en place mémoire, ou lors d'un problème de communication.

Les événements système sont gérés par des messages système qui sont transmis au superviseur. Les événements extérieurs et les événements de résolution concernent le contenu de la base, ils sont définis par des *configurations de contrôle*. Le mécanisme d'évaluation de la base qui gère la validité des configurations est utilisé pour détecter dans la base toutes les situations qui intéressent la couche réactive du contrôle. Un événement défini à partir d'une configuration de contrôle est constitué de l'identificateur de cette configuration. Aussi, pour une configuration qui devient n fois valide au cours d'un cycle d'inférence, l'événement qui correspond à cette validité sera transmis n fois au superviseur.

Transitions L'arbre des classes de situations a été introduit afin d'alléger la phase de définition des transitions : en associant une transition à un nœud de l'arbre des classes de transitions, par héritage, cette transition est définie pour toutes les feuilles inférieures à ce nœud.

$\diamond$ **Définition** *Transition*: Une transition est un triplet $t = (s_1, s_2, e)$ où $s_1 \in S$ est la classe de situations initiale et $s_2 \in \mathcal{S}_e$ est l'état final, et où e est l'événement qui provoque le changement d'état de s_1 vers s_2. L'ensemble des transitions ainsi définies est noté $\mathcal{T}$. L'événement d'une transition t est noté $Ev(t)$.

On note $As(s)$ l'ensemble des transitions qui sont définies avec s comme état initial : $As(s) = \{t \in \mathcal{T} / t = (s, s_2, e)\}$

Un ensemble de transitions $\mathcal{T}$ pour être valide sur l'ensemble de classes $\mathcal{S}_e$ et sur l'ensemble d'événements $\mathcal{E}$ ne doit associer à une classe s qu'une seule transition pour l'événement e :

$$\forall t_1, t_2 \in As(s) \; [(t_1 = (s, s_f, e) \wedge t_2 = (s, s'_f, e))] \Longrightarrow (t_1 = t_2)$$

Automate d'états finis A partir de la définition des transitions sur l'ensemble des classes de situations S, il est possible d'obtenir un automate d'états finis. La relation d'ordre partiel sur S a été introduite pour permettre une définition incrémentale des transitions par rapport aux événements. Si pour un état donné, aucune transition n'a été spécifiée pour un événement e, alors la transition associée à cet événement pour une classe supérieure sera exploitée. Cela permet de définir des transitions très générales. Par exemple les transitions associées à la classe s_{Top} concernent potentiellement toutes les situations possibles de la résolution. La définition des transitions tirables à partir d'un état de l'automate doit prendre en compte le problème des conflits entre transitions.

Pour définir les transitions tirables à partir d'un état de l'automate, on introduit l'opérateur $\oplus$ qui construit l'union de deux ensembles de transitions en résolvant les conflits au bénéfice de l'opérande gauche :

$$As(s_i) \oplus As(s_j) = As(s_i) \bigcup$$
$$\{t = (s_i, s_2, e)/t' = (s_j, s_2, e) \in As(s_j) et \forall t' \in As(s_i) : Ev(t) \neq Ev(t')\}$$

L'ensemble des transitions tirables à partir d'un état s est noté $T(s)$ et il est construit par applications successives de l'opérateur $\oplus$ (avec priorité à droite) sur les ensembles de transitions qui sont associés à toutes les classes supérieures :

$$T(s) = \oplus_{s \sqsubseteq s'}^{s' = s_{Top}} As(s')$$

Cette définition des transitions tirables à partir d'un état assure qu'à un état de l'automate ne correspond qu'une seule transition pour un événement donné. L'ensemble fini d'états $\mathcal{S}$, muni de cette définition des transitions tirables pour un état, constitue donc un automate d'états finis.

Ordre sur les transitions Le cycle d'inférence de TRAM est asynchrone : les événements ne sont pas pris en compte immédiatement mais à une étape particulière, donc après un intervalle de temps non nul. Dès lors, plusieurs événements peuvent être perçus simultanément et les transitions qui leur sont respectivement associées dans l'état courant sont en conflit. Il est donc nécessaire d'ordonner l'ensemble des transitions tirables à partir d'un état de l'automate. Cet ordre doit être total pour résoudre tous les conflits possibles.

Plusieurs approches peuvent être utilisées pour ordonner les transitions :

- L'ordre peut être défini sur l'ensemble de tous les événements. L'approche $(\mathcal{E}, \leq_{\mathcal{E}})$ interdit de relativiser l'importance d'un événement en fonction de l'état de l'automate.

- L'ordre ne peut être total sur les transitions, car des transitions qui concernent des états non comparables, ne peuvent pas être comparées.

- L'ordre peut être défini sur l'ensemble des transitions associées à un état. L'approche $(As(s), \leq_{As(s)})$ est incompatible avec l'héritage des transitions, car elle ne compare pas les transitions d'un état s avec celles d'une classe supérieure.

- L'ordre peut être défini sur l'ensemble des transitions tirables à partir d'un état.

C'est donc l'approche $(T(s), \leq_{T(s)})$ qui est adoptée : la définition du comportement réactif du système passe par la définition d'un ordre total sur les ensembles de transitions tirables à partir de tous les états de l'automate.

Interface avec la couche de gestion des tâches La couche réactive a pour fonction de définir la réponse que le système doit apporter aux événements qui surviennent pendant la résolution. Elle doit donc identifier la tâche à effectuer sans indiquer comment le faire. Cette tâche est potentiellement une tâche composite qui admet une décomposition en sous-tâches. Elle peut également être élémentaire, c'est-à-dire réalisée directement par une action.

Cette sélection de la tâche principale à effectuer est obtenue à partir de l'état courant de l'automate. Il serait possible de définir une fonction pour mettre en œuvre une sélection complexe : à chaque état correspondrait un espace de tâches dans lequel cette fonction sélectionnerait ou construirait la tâche à partir du contexte courant de la résolution. Etant donné le type d'applications auquel TRAM est destiné, nous préférons que le lien entre les états de l'automate et les tâches principales soit direct et bijectif.

Donc à chaque état de l'automate qui définit le comportement réactif correspond une tâche principale (composite ou élémentaire) qui est transmise par le superviseur au gestionnaire de tâches comme tâche à réaliser.

IV.1.2 Gestionnaire de tâches

TRAM étant destiné à des domaines d'applications pour lesquels des plans d'actions existent, son rôle est de coordonner ces schémas de résolution et non de générer de nouveaux plans. La planification dans TRAM ne prend donc pas en compte la génération de plans.

Rôle du gestionnaire de tâches Le gestionnaire de tâches a la charge d'assurer la réalisation de la tâche principale qui lui a été désignée par le superviseur. Il dispose pour cela d'un dictionnaire de tâches principales qui associe à chacune l'ensemble des méthodes qui permettent de les réaliser. La gestion des tâches consiste alors à sélectionner pour la tâche principale une méthode, celle qui est la mieux adaptée à la situation de la résolution, puis à mettre en œuvre cette méthode. Le rôle du gestionnaire de tâches est donc d'effectuer les deux étapes suivantes :

1. **Etape de sélection** : sélectionner une méthode parmi celles qui sont associées à la tâche principale.

2. **Etape de mise en œuvre** : gérer l'exécution de la méthode sélectionnée.

A une tâche composite T correspondent des méthodes qui sont des plans d'actions : un plan d'actions définit la décomposition de la tâche T en sous-tâches $T_1, ..., T_4$ et le flot des entrées E_i et des sorties sorties S_j entre la tâche composite et ses sous-tâches d'une part, et entre les sous-tâches d'autre part (Figure IV.2).

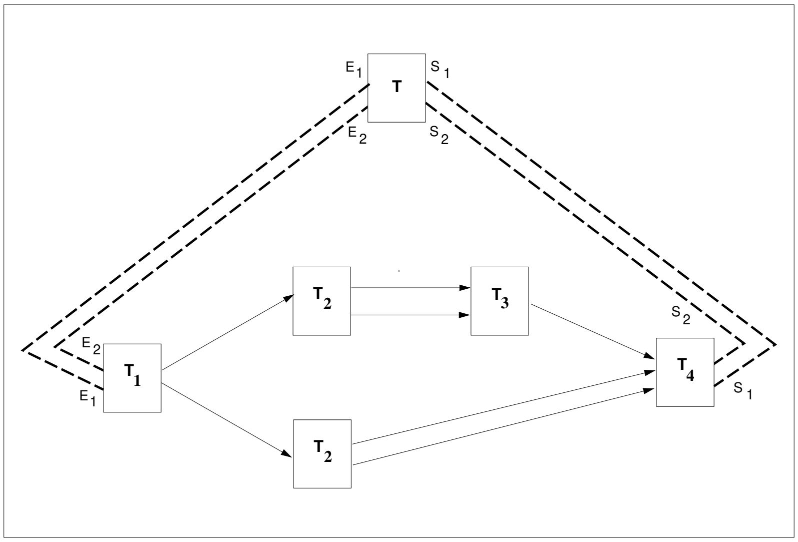

FIG. IV.2 - *Définition de la décomposition d'une tâche T et du flot des données.*

Problème de l'accès aux données de la résolution La gestion des tâches fait intervenir les données de la résolution. Alors que la couche réactive ne perçoit de la résolution que les événements qui valident les transitions, la gestion des tâches utilise les données pour, d'une part sélectionner la méthode à utiliser, et d'autre part mettre en œuvre cette méthode. Cette implication des données dans la couche de gestion des tâches est plus forte que pour la couche réactive. En effet, un événement peut être structuré comme un résumé (éventuellement limité à un identificateur) des données de la résolution, alors que l'étape de sélection doit consulter les données et l'étape de mise en œuvre doit désigner les données : elle doit identifier les entrées nécessaires à l'exécution d'une action, et elle doit pouvoir désigner les sorties pour gérer le flot des données dans un plan d'actions.

En résumé, pendant l'étape de sélection d'une méthode, le gestionnaire de tâches consulte les données, et pendant l'étape de mise en œuvre de la méthode sélectionnée, il désigne les données pour gérer le flot des entrées-sorties.

Par ailleurs les données sont contenues dans la base qui assure la modélisation de la résolution. Il faut donc, soit que le gestionnaire de tâches délègue une partie de son activité au gestionnaire de la base, soit qu'il accède au contenu de la base. Cette dernière possibilité soulève le problème de la duplication (permanente ou momentanée) des données entre la base et le gestionnaire de tâches. Il est donc préférable de mettre en œuvre dans la base une partie des fonctions du gestionnaire de tâches.

Gestionnaire de tâches et gestionnaire de la base La répartition des fonctions de planification entre le gestionnaire de tâches et le gestionnaire de la base est motivée par les contraintes liées à l'accès aux données. Les différentes étapes du traitement d'une tâche élémentaire sont les suivantes :

$$T \begin{array}{llll} \stackrel{EF}{\Longrightarrow} & (m_1 ... m_n) & \stackrel{ES}{\Longrightarrow} & m_i \\ \stackrel{EP}{\Longrightarrow} & T(x_1 ... x_p) & \stackrel{EV}{\Longrightarrow} & T(v_1 ... v_p) \end{array} \stackrel{EA}{\Longrightarrow} \quad m_i(v_1 ... v_p)$$

L'étape de formulation (EF) désigne pour la tâche les méthodes possibles pour sa réalisation. Cette étape n'est pas seulement une consultation du dictionnaire des tâches qui est associé à la bibliothèque disponible car elle fait intervenir des critères de contrôle qui affectent l'étape de sélection (ES). L'étape de désignation des paramètres d'appel de la tâche (EP) gère le flot des données par rapport au plan d'actions dans lequel se trouve la tâche. L'étape d'accès aux valeurs des paramètres (EV) prépare l'étape d'appel de la méthode sélectionnée (EA). La séparation entre le gestionnaire de tâches et le gestionnaire de la base est nécessaire : le flot des données est défini par le plan d'actions et la gestion des critères de contrôle lui est associée.

La répartition des différentes étapes entre le gestionnaire de tâches ($\mathcal{GT}$) et le gestionnaire de la base ($\mathcal{GB}$) est la suivante :

$$T \begin{array}{llll} \stackrel{\mathcal{GT}}{\Longrightarrow} & (m_1 ... m_n) & \stackrel{\mathcal{GB}}{\Longrightarrow} & m_i \\ \stackrel{\mathcal{GT}}{\Longrightarrow} & T(x_1 ... x_p) & \stackrel{\mathcal{GB}}{\Longrightarrow} & T(v_1 ... v_p) \end{array} \stackrel{\mathcal{GB}}{\Longrightarrow} \quad m_i(v_1 ... v_p)$$

Afin d'éviter la duplication des données, les étapes qui font intervenir les données sont prises en charge par le gestionnaire de la base. Les autres étapes ne nécessitent que la désignation indirecte des données, mais manipulent les plans d'actions. Elles relèvent donc bien du gestionnaire de tâches.

Pour les tâches composites, la gestion de la décomposition de la tâche en sous-tâches est une étape supplémentaire. Elle intervient lorsque la méthode m_i obtenue lors de l'étape de sélection (ES) est un plan d'actions. La gestion d'un plan d'actions met en œuvre des algorithmes classiques, mais soulève le problème de la gestion du flot des données. Un plan d'actions ne spécifie pas seulement les instructions de contrôle (séquencement, itération, test) mais également la gestion des entrées-sorties entre les sous-tâches.

La désignation des données pour le flot des entrées-sorties entre les tâches est faite au moyen du numéro de cycle (une seule tâche est réalisée à chaque cycle) et de la position du paramètre dans les entrées-sorties de la tâche.

IV.2 Réalisation : approche client/serveur

Du point de vue de sa réalisation, TRAM est une architecture de type client/serveur qui permet de faire coopérer entre eux des codes de calcul indépendants.

"Est appelé client-serveur un modèle de fonctionnement logiciel dans lequel plusieurs programmes autonomes communiquent entre eux par échange de messages au travers d'un réseau et où certains programmes offrent des services qui sont utilisés par les autres [Sinh92]". Le programme qui offre le service est appelé serveur, celui qui l'utilise, client. L'utilisation abondante de cette technique correspond au besoin de faire coopérer des systèmes logiciels indépendants. TRAM a pour vocation de constituer une plate-forme logicielle qui permet d'intégrer des modules algorithmiques indépendants pour les utiliser en résolution de problèmes. Il est donc concerné par cette approche.

IV.2.1 Motivation de l'approche client/serveur

La principale contrainte imposée à TRAM (voir I.5.2) concerne la réutilisabilité. Outre les plans d'actions, il est nécessaire d'intégrer sans modification les programmes que l'outil doit piloter. Or ces programmes interagissent avec la base pour obtenir les valeurs de leurs paramètres et pour fournir celles de leurs résultats. Il est nécessaire que ces interactions soient gérées sans modifier les programmes externes.

Du point de vue de la réalisation, cette contrainte exclut donc la possibilité d'intégrer dans ces programmes les fonctions de gestion des accès à la base. Dès lors, c'est le gestionnaire de la base qui assure les transferts de données entre la base et un module externe. Cette approche correspond ainsi à un fonctionnement de type client-serveur.

Plusieurs réalisations sont possibles pour un fonctionnement de type client-serveur :

- présentation distribuée entre le serveur et le client,
- présentation déportée sur le client,
- logique d'application distribuée sur le client et le serveur,
- gestion des données déportée sur le serveur,
- gestion des données distribuées sur le client et sur le serveur.

L'organisation du système TRAM et des modules algorithmiques ne correspond pas seulement à la gestion déportée des données, car l'application est distribuée entre les modules externes et TRAM qui assure le contrôle de l'application.

IV.2.2 Spécification du comportement des modules

Pour pouvoir être intégrés dans l'architecture client-serveur de TRAM, les modules algorithmiques sont considérés comme des boîtes noires dont le comportement vis-à-vis de TRAM est spécifié par une carte de description.

Problèmes

La spécification du comportement de modules algorithmiques pose plusieurs problèmes [Chai92b] :

Interactions limitées Le comportement d'un module algorithmique n'est spécifié que par rapport au système TRAM, c'est-à-dire par rapport au contenu de la base. Pour que les mises à jour de la base effectuées à partir de l'exécution d'un module soient spécifiables, il faut que les interactions entre le module et la base soient limitées. Or ces interactions sont effectuées à partir des entrées et des sorties du module. Il est donc nécessaire que la signature d'un module soit clairement identifiée, c'est-à-dire que le format de ses paramètres et de ses résultats soit fixé et connu par avance.

Cette contrainte est forte, mais compatible avec les modules algorithmiques utilisés en informatique scientifique et technique.

Indépendance des structures de données Les modules intégrés dans le système TRAM comportent des structures de données qui leur sont propres et qui concernent entre autres leurs paramètres et leurs résultats. Pour garantir l'indépendance entre ces modules et TRAM, il est nécessaire que ces structures de données puissent être modifiées sans que cela n'affecte ce dernier. Il faut donc que le langage de spécification soit polymorphe par rapport aux types des paramètres et des résultats.

La transmission des données entre le gestionnaire de la base et les serveurs de traitement sur lesquels s'exécutent les modules algorithmiques est gérée par un protocole basé sur la taille de ces données. Pour TRAM, toutes les structures de données définies dans les modules externes sont caractérisées en taille, et en position relative pour les champs qu'elles contiennent. Cette caractérisation est faite à partir des fichiers de définition des structures, elle est ensuite chargée dans le gestionnaire de la base. La gestion des structures de données externes constitue donc une étape de nature déclarative.

Gestion du nombre variable d'arguments En informatique scientifique et technique, les paramètres ou les résultats d'un module algorithmique sont généralement fournis sous la forme d'une liste de données dont la cardinalité varie dynamiquement. Il est donc nécessaire de prendre en compte cet aspect. Par exemple, en contrôle non destructif, la caractérisation d'une pièce métallique aboutit à l'obtention d'une liste de défauts dont le nombre est imprévisible.

Pour aborder ce problème, nous avons adopté la même solution que celle qui est utilisée pour spécifier la signature d'une fonction à nombre variable d'arguments dans les bibliothèques de calcul : un protocole permet de gérer d'une part le type commun aux données de la liste et d'autre part le nombre d'éléments. Le type des données est tout d'abord spécifié. Puis le protocole transmet le nombre d'arguments avant les arguments eux-mêmes, ou il identifie la fin de la liste des arguments par un élément spécial.

Gestion du type variable d'un résultat Les résultats doivent être spécifiés par rapport à la modélisation que TRAM gère pour les entités qu'ils concernent. Cette spécification doit d'une part rendre compte du type concret (entier, structure) pour que les contrôles puissent être effectués, et d'autre part de son concept et si possible de la classe à laquelle il doit être rattaché.

Cette contrainte est forte, elle oblige à ne définir chacun des éléments de la signature d'un module que par un seul concept. Or les modules algorithmiques manipulent des données indépendamment de leur sémantique externe et un module peut être utilisé dans des contextes différents pour calculer des matrices qui sont respectivement associées à un champ magnétique d'une part, et à un nuage de points d'autre part. L'argument d'un module ne pouvant être spécifié que par rapport à un seul concept, il est nécessaire de définir deux actions différentes pour le même module algorithmique : l'une qui intervient en électromagnétisme et l'autre en analyse de données. Ainsi, pour chaque utilisation d'un module algorithmique, on définit une action différente qui constitue la méthode de la tâche qu'elle permet de réaliser.

Si pour une action donnée, chaque argument est associé à un concept, il est nécessaire de spécifier également sa classe d'appartenance lors de la création d'une instance. Par exemple, l'analyse électromagnétique d'un défaut produit trois matrices qui décrivent ce défaut. Les propriétés de ces matrices ne peuvent être connues par avance, elles dépendent des caractéristiques de la pièce métallique qui est examinée. En spécifiant la création d'une instance dans la classe racine de son concept, on laisse au mécanisme de classification la charge de déterminer les propriétés de cette instance et donc les classes auxquelles elle appartient. Ces classes d'appartenance peuvent alors rendre activables des actions qui ne l'étaient pas tant que cette instance n'était rattachée qu'à la classe racine du concept.

Le mécanisme de classification d'instances joue ainsi le rôle de la détermination du type des résultats des actions par rapport à la modélisation qui est faite de la résolution.

Eléments de la spécification d'un module algorithmique

La carte de description des modules algorithmiques spécifie leur comportement par rapport à TRAM. Elle est composée des éléments suivants :

1. La **condition d'activation** décrit les états de la base qui rendent activable le module algorithmique.

2. Le **format des paramètres** décrit la façon dont les valeurs des paramètres d'appel sont obtenus à partir du contexte d'activation défini par la condition d'activation.

3. Le **format des résultats** spécifie le type et la cardinalité des résultats du module.

4. L'**appel du programme** spécifie le nom du programme et décrit ses arguments.

5. La **proposition** décrit les mises à jour de la base qui sont faites en utilisant les valeurs des résultats du module.

6. Le **test d'erreur** décrit la procédure de test des valeurs des résultats qui indique si le module a renvoyé un code d'erreur.

7. L'**action d'erreur** décrit les mises à jour de la base qui doivent être effectuées lors du cas d'erreur.

8. L'**identificateur du serveur** désigne le serveur de traitement sur lequel ce module s'exécute.

Exemples de cartes de description de modules algorithmiques Ces exemples sont extraits du projet AIRES (voir IV.3.2).

Le module d'inspection électromagnétique active le capteur et crée des données d'inspection. Sa carte de description est la suivante :

```
server          = 1 ;
condition       = exist X in CP-emsensor ;
                / X.state = ready ;
program         = EM-scanning(&X.struct,&X.location,a3Ddata,&error) ;
parameters      = <X.struct : SENSORPARAMETERS, X.location : POSITION > ;
results         = < a3Ddata : BVALUES[NBBVALUES], error : short > ;
proposition     = create Y in CP-em3Ddata
                / Y.with-sensor = X,
                  Y.struct = a3Ddata,
                  Y.pktr-found = no,
                  Y.identified = no,
                  Y.nn-classified = no;
                  modify X
                / X.state = error ;
error-test      = errorcode <> 0 ;
error-action    = create Z in CP-emerror
                / Z.state = errorhasoccured,
                  Z.code = error ;
```

Carte de description du module d'inspection électromagnétique (Projet AIRES).

Il est possible de définir des actions qui ne correspondent à aucun module algorithmique, mais qui ne font que mettre à jour la base. Par exemple, à la fin de l'inspection par électromagnétisme, une action est déclenchée pour clore l'opération d'inspection :

```
server          = 1 ;
condition       = exist X in CP-emsensor ;
                / X.another-location = no ;
                / X.state = done ;
                = exist Y in CP-emflatplate
                / Y.state = scaninprogress ;
program         = NO-PROGRAMM ;
parameters      = ;
results         = ;
proposition     = modify Y
                / Y.state = scanfinished ;
error-test      = ;
error-action    = ;
```

Carte de description d'une action sans module algorithmique qui détecte la fin de l'analyse de l'échantillon (Projet AIRES).

IV.2.3 Réalisation logicielle

TRAM est écrit en langage C, il constitue une architecture logicielle multi-processus, modulaire et asynchrone : le superviseur, le gestionnaire de tâches, le gestionnaire de la base et les serveurs de traitements constituent chacun des processus autonomes qui peuvent s'exécuter sur différents processeurs et communiquer via le réseau utilisé. Deux architectures sont actuellement opérationnelles :

1. Une architecture UNIX sur stations de travail reliées par un réseau Ethernet est dédiée au calcul scientifique. Les modules sont distribués sur différentes stations de travail, chaque module disposant ainsi de sa propre interface. Les communications sont mises en œuvre à partir des fonctions TLI (Transport Level Interface).

2. Une architecture Helios sur carte de Transputers peut être embarquée pour les applications de robotique autonome mobile. Les communications sont assurées par le réseau des transputers.

La réalisation de TRAM a été menée par plusieurs personnes en ce qui concerne les différents composants du logiciel : M. Antoniakos dans le cadre d'une thèse CNAM (superviseur et gestionnaire de tâches, projet AIRES), M. Chaillot (base d'objets), E. Crochon (architecture HELIOS, projet TRANS-TRAM II), A. Koenig (architecture UNIX, projet AIRES), L. Glénaz (interface TRAM), G. Gonon (analyseurs syntaxiques, générateurs de code et communications TLI), S. Guédon dans le cadre d'une thèse de doctorat (projet SIGMA).

IV.3 Applications

Nous décrivons dans cette section la définition d'une application et les applications qui permettent de valider notre architecture TRAM. Actuellement trois projets sont en cours.

IV.3.1 Etapes de définition d'une application

La réalisation est composée de quatre étapes qui sont essentiellement de nature déclarative, limitant la phase de programmation à l'intégration des programmes pilotés dans les structures d'accueil. La liste qui suit explicite ces différentes étapes, mais ne constitue pas une méthodologie pour la conception de l'application. En particulier, l'identification des concepts et des classes, ainsi que la spécification des tâches et du comportement réactif restent à la charge du concepteur de l'application, et nous ne sommes pas (encore) en mesure d'identifier l'apport possible des différentes méthodologies proposées (KADS, KOD) pour l'aider dans ce travail.

1. **Définition de la base**
   La définition de la base consiste à construire l'ensemble des concepts et de leurs classes. Plusieurs bases peuvent être définies.

2. **Définition du comportement réactif**
   La définition du comportement réactif intervient au niveau de la couche réactive et au niveau de la gestion de la base.

   (a) **Automate d'états finis**
   La définition de l'automate a été donnée en IV.1.1. Elle consiste à identifier les états de l'automate, les tâches qui leur sont associées et les transitions qui permettent de passer d'un état à l'autre.

   (b) **Configurations de contrôle**
   La définition des configurations de contrôle spécifie les événements qui doivent être détectés dans la base et transmis au superviseur.

3. **Définition des tâches et de leurs méthodes**
   La définition des plans qui spécifient les décompositions des tâches est faite à partir d'un éditeur de plans analogue au module *Cantata* de *Khoros* [Will92] ou à l'outil *ICE* développé dans le cadre du projet *VIDIMUS* [1]. La définition des méthodes associées aux tâches fournit pour chaque tâche la hiérarchie des identificateurs de ses méthodes plans ou modules algorithmiques.

4. **Spécification des modules algorithmiques**
   La spécification du comportement des modules définit leurs cartes descriptives. A partir d'une carte descriptive, la configuration d'activation d'un module et la description de ses entrées-sorties sont obtenues.

5. **Intégration logicielle des modules algorithmiques**
   L'intégration logicielle des modules algorithmiques dans l'outil TRAM consiste à construire le squelette logiciel qui répartit les modules algorithmiques sur les différents serveurs de traitement. Cette étape fournit plusieurs fichiers qui sont utilisés pour générer le code exécutable de l'application.

IV.3.2 Contrôle non destructif : projet AIRES

AIRES est un projet européen [2] en contrôle non destructif. Le système est opérationnel chez nos partenaires : Non Destructive Control Center of University College of London et Deutsch Aerospace.

[1] Projet ESPRIT 2592 VIDIMUS

[2] BRITE-EURAM AIRES CT91-0464

Le rôle de TRAM est d'organiser la coopération de capteurs pour détecter et caractériser les défauts de pièces métalliques. Une caméra (capteur vision) fournit les images de la pièce à partir desquelles un capteur électromagnétique est positionné pour analyser localement le matériau métallique. Le capteur électromagnétique utilise les courants induits pour obtenir un signal qui permet de reconstituer la structure du matériau. Ce signal permet de caractériser les défauts qui avaient éventuellement été détectés par le capteur vision.

Dans son état actuel, AIRES ne pose que peu de conflits pour les tâches élémentaires, mais oppose plusieurs phases possibles de résolution au moyen du capteur vision, du capteur électromagnétique ou des deux capteurs simultanément.

Le diagramme d'états défini pour spécifier le comportement réactif de l'application AIRES est donné en figure IV.3.

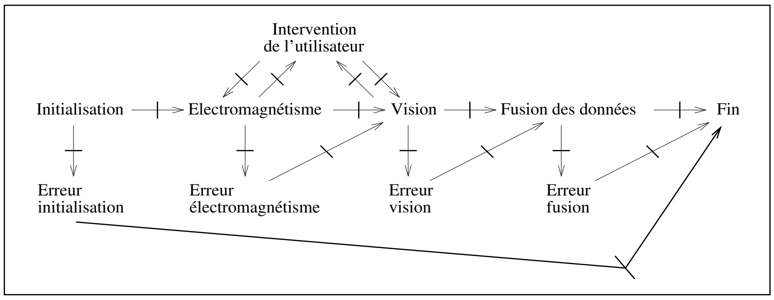

FIG. IV.3 - *Automate d'états finis de l'application AIRES.*

Les concepts de la base décrivent respectivement les pièces contrôlées (géométrie, statut), les capteurs (paramètres, état, position, spécimen inspecté), les données d'inspection (données en trois dimensions du champ magnétique, image de vision), les caractéristiques (défauts, type défaut) et les erreurs.

Les actions sont au nombre de 36. Pour la partie électromagnétique, il s'agit de l'initialisation du capteur, de l'inspection de la pièce, de l'identification des caractéristiques, de l'analyse du champ magnétique, de la classification, comparaison et caractérisation des défauts détectés, de la terminaison et de la remise à zéro du capteur. Pour la partie vision, il s'agit de l'initialisation du capteur, du positionnement du capteur, de l'acquisition d'une image, de l'analyse d'une image, des règlages du capteur (lumière, focus), de l'élimination du fond, des différents traitements de l'image, de l'extraction des caractéristiques et de la préparation des données de fusion.

AIRES utilise la version Unix de TRAM.

IV.3.3 Détection sous-marine : projet SIGMA

Dans le cadre d'une thèse, S. Guédon utilise TRAM pour son travail en détection sous-marine [Gué93].

Le problème est lié à l'exploitation de données issues d'un réseau de capteurs de type magnétomètre RMN (Résonance Magnétique Nucléaire) qui est disposé dans la zone de mer surveillée. Un signal utile étant détecté, il s'agit de piloter des modules algorithmiques en traitement du signal pour extraire l'information utile. Le problème est complexe, il fait intervenir les signaux magnétiques spécifiques de l'environnement géographique, les caractéristiques géométriques du réseau de capteurs et du bruit.

Le travail décrit ici ne s'attache qu'à une partie du problème. Cinq tâches principales ont été identifiées, admettant chacune au moins une décomposition en sous-tâches. Ces cinq tâches sont :

1. La tâche d'initialisation prend en compte l'information utile pour la résolution du problème.
2. La tâche d'acquisition des données à traiter met à jour l'horizon d'observation.
3. La tâche d'apprentissage des bruits à supposer l'absence d'anomalie magnétique pour définir l'image "à blanc" du site.
4. La tâche de détection d'un signal utile analyse une anomalie magnétique.
5. La tâche d'aide à la décision analyse le signal utile.

Dans sa phase actuelle, la base est consituée de 12 concepts qui décrivent l'environnement géographique, le réseau de capteurs, le signal observé. Les modules algorithmiques concernent l'initialiation et la configuration du réseau, l'acquisition et le traitement des mesures, l'analyse intercapteurs et les procédures de décision.

Cette application est particulièrement intéressante, car ses caractéristiques correspondent bien à celles des applications visées par TRAM : plusieurs modules algorithmiques sont en conflit pour réaliser des tâches élémentaires, plusieurs états désignent des tâches principales dont les décompositions en soustâches sont complexes, et surtout, les volumes de données sont importants, ce qui met la gestion de la base à rude épreuve. La version utilisée est la version Unix de TRAM.

IV.3.4 Robotique autonome mobile : projet TRANSTRAM II

TRANSTRAM II [3] est un projet avec la Sagem comme partenaire; il consiste à utiliser le système TRAM pour définir l'architecture de contrôle d'un robot autonome mobile [Croc93].

Le système de navigation d'un robot autonome mobile est chargé d'assurer la bonne exécution de la mission confiée au robot. Pour être fiable, il doit pouvoir assurer une réponse rapide à tous les événements qui sont répertoriés :

[3] Projet MRE N91-P-0194 TRANSTRAM II

les événements extérieurs caractérisés par les évolutions de l'environnement ou par une intervention de l'opérateur pour modifier la mission ou le mode opératoire, les événements système provenant des erreurs de fonctionnement des ressources, et les événements de navigation qui sont liés aux problèmes lors de l'exécution de la mission.

Les objectifs de l'utilisation de TRAM sont au nombre de trois :

1. Les architectures de contrôle robotique utilisées au LETI sont des systèmes réactifs, hiérarchisés et distribués. Plusieurs modules sont affectés au contrôle des différents éléments du robot, depuis la gestion de la mission jusqu'aux capteurs et aux effectueurs. TRAM intervient pour intégrer un système de contrôle réactif centralisé qui est connecté directement avec les différents niveaux du système de contrôle. Si la mission se déroule correctement, TRAM n'intervient pas, sa base est seulement mise à jour pour stocker des informations synthétiques relatives au fonctionnement du robot. En cas d'événements imprévus, le rôle de TRAM est d'accroître la réactivité du système de contrôle en intervenant plus rapidement que le système de contrôle hiérarchique qui perd du temps en communication.

2. La gestion des tâches assurée par TRAM permet d'expliciter les stratégies de contrôle du robot sous la forme de combinaisons d'actions qui sont réalisées suivant l'évolution du contexte.

3. L'intégration de l'existant est également présente dans les exigences du développement d'un système de contrôle robotique. En effet, il est intéressant de pouvoir intégrer sans modification des modules opérationnels comme les modules de traitement associés aux capteurs. TRAM joue alors le rôle de structure d'accueil pour le contrôle d'exécution de ces modules externes.

Le couplage de TRAM avec un système de contrôle hiérarchisé est opérationnel dans le projet ESPRIT PANORAMA [4], mais seulement au niveau logiciel, TRAM n'intervient pas encore dans le déroulement de la mission.

Les projets de robotique utilisent la version embarquée de TRAM sur carte de Transputers.

IV.4 Perspectives pour l'explication

Bien que l'explication ne constitue pas le thème de notre travail, nous avons porté un intérêt tout particulier à la littérature qui concerne le sujet. La conception de TRAM a été guidée par le souci de définir un système explicable, dans le sens où il satisfait les exigences liées à la production d'explications. Cette section décrit notre proposition pour l'explication dans un $\mathcal{ERP}$ réactif.

[4] ESPRIT II N 2483 PANORAMA

L'explication de la résolution est un aspect important des systèmes à base de connaissances en général et des $\mathcal{ERP}$ en particulier. Son rôle est de fournir à l'utilisateur les informations qui lui permettent d'évaluer, de comprendre, de se former, de "debugger"... Cette abondance d'objectifs est confrontée à la difficulté de produire une explication convaincante. Aussi nous adoptons une approche pragmatique qui consiste à définir un système d'information de type hypertexte, mis à la disposition de l'utilisateur pour construire l'explication qui lui convient. Cette structure hypertexte associe la trace à la base de connaissances explicatives, c'est pourquoi nous l'appelons une hypertrace.

IV.4.1 Problématique de l'explication

La problématique de l'explication en résolution de problèmes est complexe et n'admet pas encore de solution satisfaisante et acceptée par tous. Elle constitue un thème de recherche étroitement lié à celui de la conception des systèmes à base de connaissances. Cette section présente les éléments de l'explication.

Définitions et objectifs

La définition de l'explication n'est pas disponible dans une forme reconnue, elle est souvent posée en termes d'objectifs. De façon générale, l'explication fait intervenir trois éléments qui sont : un système de résolution de problèmes, une résolution de problème achevée ou en cours et un utilisateur. L'explication est un dialogue [Bala90], aussi nous préférons utiliser le terme d'interlocuteur pour désigner celui à qui est destinée l'explication, réservant le terme d'utilisateur pour le système de résolution.

Dans ce contexte, l'explication est définie comme la fourniture à l'interlocuteur d'informations intelligibles sur le déroulement de la résolution. Le terme d'intelligible spécifie que ces informations sont exploitables par l'interlocuteur. Le fait que l'explication soit un dialogue confère à sa présentation un rôle important.

Il s'agit donc d'exposer, d'enseigner, de justifier, de faire comprendre les éléments de la résolution du problème.

Etapes importantes

A partir de ce vaste problème, différents travaux ont été menés sur l'explication conjointement au développement des systèmes à base de connaissances. Plusieurs grandes étapes sont intervenues :

- **MYCIN - Shortliffe - 1974** MYCIN est un système expert à base de règles développé pour des applications médicales [Shor74]. Ce système est le premier à mettre en avant une explication de la résolution. Cette explication consiste à utiliser l'arborescence des buts pour répondre à des questions du type *Pourquoi?*, *Comment?* en exhibant les règles qui ont été déclenchées.

Davis [Davi80] a étendu cette explication en construisant le graphe complet du déclenchement des règles et en parcourant le graphe pour rechercher les règles qui concernent la question posée par l'interlocuteur.

- **NEOMYCIN - Clancey - 1981** La critique de l'explication produite dans MYCIN est liée à la nature non structurée de l'ensemble des règles qui limite le contenu possible de l'explication. W. Clancey a utilisé le système MYCIN pour enseigner la médecine et a conçu le système GUIDON. Il a montré que des connaissances stratégiques étaient cachées dans le contenu des règles et donc inaccessibles à l'explication.

  Dès lors, il a imaginé de regrouper ces connaissances stratégiques dans des règles spécifiques ("meta-règles") qui lui permettent de structurer la base de règles. NEOMYCIN [Clan81] est le premier système dont la conception ait été motivée par l'explication. Il apparaît donc que la mise en œuvre de l'explication impose des exigences au système expliqué.

- **XPLAIN - Swartout - 1981** Le système XPLAIN [Swar83] est un système expert en diagnostic médical. Il est basé sur la génération automatique de programmes à partir des expériences précédentes. A partir des données ainsi collectées, il utilise un "modèle causal" pour la théorie du domaine et un ensemble de "principes du domaine" pour acquérir les "connaissances de surface". Les "connaissances de surface" sont des heuristiques qui permettent de résoudre des problèmes simples par opposition aux "connaissances profondes" qui utilisent des modèles du domaine pour résoudre des problèmes plus complexes. Les "connaissances de surface" peuvent être justifiées par les "connaissances profondes" qui ont participé à leur génération.

- **Systèmes experts explicables**

  A partir du système NEOMYCIN qui a explicité les "connaissances de stratégie" et du système XPLAIN qui a positionné les "connaissances de surface" par rapport aux "connaissances profondes", une famille de systèmes experts explicables s'est constituée, préconisant la fourniture d'informations supplémentaires (inutiles pour la résolution) lors de la conception d'un système expert pour être utilisées par l'explication. B. Chandrasekaran a identifié plusieurs niveaux de connaissances dans un système à base de connaissances et, pour faire intervenir une modélisation de plus haut niveau que ne le permettent les règles, il a proposé les tâches génériques [Chan86] puis la structure de tâche [Chan92]. Les systèmes experts "de seconde génération" ont ainsi organisé les connaissances en deux catégories, étiquettées respectivement par les termes "profondes" et "de surface". Pour la comparaison des principaux systèmes explicatifs, nous renvoyons à [Moor88] et [Abu90].

Directions de recherche

A partir de ces travaux, plusieurs directions de recherche sont suivies pour aborder le problème de l'explication dans les systèmes à base de connaissances.

- **Acquisition des connaissances** L'analyse des systèmes existants montre que l'information qu'ils contiennent permet de résoudre les problèmes mais ne suffit pas à l'explication. Les connaissances compilées et les connaissances manquantes limitent l'explication. A partir de ce constat, une direction de recherche importante consiste à améliorer les techniques d'acquisition des connaissances avec comme objectif la conception de systèmes explicables, c'est-à-dire qui contiennent plus d'information pour l'explication. Les tâches rendent possible une modélisation de haut niveau des connaissances de résolution. Aussi les différentes approches en acquisition des connaissances utilisent-elles la notion de tâche [Dien90].

- **Présentation de l'explication** La présentation de l'explication joue un rôle important, elle conditionne la qualité de l'explication. Plusieurs approches sont envisagées pour présenter l'explication : les textes prédéfinis, la génération du langage naturel [Bala90], les graphes conceptuels, les structures hypertextes.

- **Modélisation de l'interlocuteur** Le rôle de l'interlocuteur a été identifié dès le début de l'explication dans les systèmes à base de connaissances. Plusieurs types d'interlocuteurs sont possibles. On distingue notament l'expert du domaine, l'étudiant, le concepteur du système, l'utilisateur du système. Pour adapter la qualité de l'explication à l'interlocuteur, il est donc intéressant de disposer d'un modèle de cet interlocuteur. La définition de ce modèle soulève des difficultés évidentes et n'admet pas encore (à notre connaissance) de solution convaincante.

- **Architectures explicables** L'explication intervient dès la conception du système à base de connaissances dans lequel on souhaite la mettre en œuvre. En explicitant les motivations des caractéristiques d'une architecture lors de sa conception, on identifie des informations qui seraient perdues autrement. De même, en imposant au système les contraintes de l'explication, on augmente ses qualités vis-à-vis de l'explication [Swar91]. De plus, pour permettre une explication satisfaisante, il faut que l'activité de résolution qui est menée par le système soit explicitée.

  Ces différentes exigences tendent à définir les caractéristiques des systèmes à base de connaissances explicables, dans le sens où ils peuvent supporter la mise en œuvre de l'explication. L'explication est alors perçue comme un problème à part entière [Abu90] qui opère sur la trace de la résolution. Parmi les caractéristiques des systèmes explicables, la modélisation de haut niveau que fournissent les tâches contribue à améliorer l'explicabilité [Davi90].

IV.4.2 Approche choisie

Face à ce statut de l'explication dans les systèmes à base de connaissances, qui identifie les problèmes sans les résoudre, nous adoptons une approche résolument pragmatique. TRAM met en œuvre une modélisation des connaissances basées sur les tâches et sur la discrétisation des états possibles de la résolution. A partir de ces éléments utiles pour l'explication, notre approche consiste à fournir à l'utilisateur les moyens de construire lui-même son explication.

Explication et traitement de l'information

L'approche que nous avons choisie correspond à celle de M. Wick. Il définit l'explication dans le cadre du traitement de l'information : "*L'explication est un procédé de traitement de l'information qui prend en entrée une opération de traitement de l'information, et qui fournit en sortie une description de cette opération de traitement de l'information*" [Wick92].

Selon cette définition, l'explication qui est associée à la résolution d'un problème n'est pas nécessairement fidèle à cette résolution. Elle peut reconstruire complètement la résolution pour exhiber une description plus pertinente. Cette définition de l'explication est compatible avec l'importance qui a été attribuée à l'interlocuteur de l'explication : la description est une reconstruction de la résolution, ce qui laisse toute liberté de l'adapter aux besoins et au niveau de l'interlocuteur.

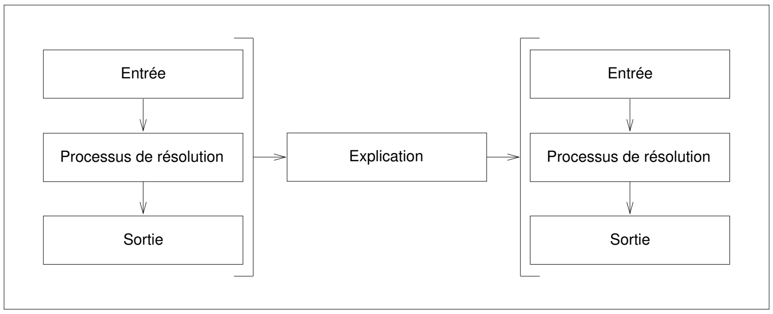

FIG. IV.4 - *Explication comme processus de traitement de l'information*

L'opération de reconstruction de la résolution pour obtenir l'explication met en évidence le rôle de la trace de la résolution expliquée. En effet, les éléments nécessaires à l'explication sont ceux qui interviennent dans l'opération de traitement de l'information qui construit la description de la résolution. Ces éléments sont l'entrée de la résolution, l'activité de la résolution et sa sortie

(Figure IV.4). Le fait que l'activité de la résolution soit une entrée de l'opération d'explication identifie le rôle de la trace et donc réhabilite les explications basées sur la trace.

Nature de la trace

La trace de la résolution étant identifiée comme l'entrée de l'explication, il faut expliciter sa nature. W. Clancey a récemment proposé une description abstraite de la résolution dans les systèmes à base de connaissances [Clan92]. Il qualifie cette approche de *"modélisation qualitative de processus"*. La modélisation des connaissances est organisée en un graphe plus ou moins complexe, et une inférence consiste à appliquer des opérateurs de construction au graphe initial pour l'étendre. W. Clancey justifie longuement cette description abstraite en examinant plusieurs systèmes afin d'identifier quelques opérateurs.

Cette description de l'activité de résolution dans un système à base de connaissances sous la forme d'un construction de graphe identifie la nature de la trace : la trace est la suite des graphes construits par applications successives des opérateurs mis en œuvre pendant la résolution.

L'entrée du processus explicatif est donc un graphe qui représente tous les états successifs de la modélisation des connaissances pendant la résolution. L'opération de traitement de l'information qui construit l'explication consiste donc à utiliser ce graphe pour obtenir une description de la résolution. Cette description peut être également perçue comme un graphe. Dans ce cas, l'explication de la résolution est une opération de transformation de graphes.

Trace et base de connaissances explicatives

La trace contient l'historique de l'activité du système de résolution, et toute étape de la résolution qui n'y figure pas ne sera pas disponible pour l'explication. Cependant, la trace ne contient pas toute l'information nécessaire à l'explication. Il faut donc lui associer une base de connaissances explicatives.

Les connaissances explicatives sont de différentes natures [Moor88] : les connaissances explicatives du domaine fournissent les définitions et les descriptions des éléments qui apparaissent dans la trace; les connaissances explicatives actives contiennent les stratégies et les principes qui sont utilisés pour manipuler les connaissances explicatives du domaine. La conception d'un système explicatif requiert donc la construction de la base de connaissances explicatives qui est associée au système expliqué.

Nous considérons dans ce qui suit que les connaissances explicatives du domaine sont des documents qui expliquent de façon statique les éléments de la résolution. Nous n'explicitons pas la nature de ces documents.

Présentation hypertexte de l'explication

L'état de l'art sur l'explication a identifié l'explication comme un dialogue entre le système explicatif et l'interlocuteur, d'où l'importance qui est attribuée à la présentation et à la modélisation de l'interlocuteur. Or les différentes modélisations de l'interlocuteur qui ont été proposées dans la littérature sur l'explication ne sont pas convaincantes : la modélisation sous la forme d'un vecteur de variables numériques, qui indiquent le degré de connaissance dans un domaine, est insuffisante.

Face à l'échec de la modélisation de l'interlocuteur, et aux difficultés liées à la présentation (langage naturel), une solution possible consiste à laisser l'interlocuteur construire sa propre explication en lui fournissant les outils et l'information nécessaires. L'information disponible est contenue dans la trace et dans la base de connaissances explicatives qui est associée à la résolution. Cette information est organisée sous la forme de graphes. Il est donc naturel de choisir les systèmes hypertextes comme outil de présentation.

"Un système hypertexte est un système capable de gérer un ensemble d'informations par accès non séquentiels. Il consiste en un réseau de nœuds reliés entre eux par des liens logiques." [Luca90]. La variété des nœuds et des liens rend ce type de systèmes très souple. L'utilisateur d'un système hypertexte est libre d'explorer le réseau qui lui permet d'accéder aux informations.

IV.4.3 Hypertrace dans un système réactif

Nous proposons ci-dessous la mise en œuvre d'un système d'information basé sur une structure hypertexte pour expliquer la résolution dans un système réactif.

Système d'informations hypertexte

D. Lucarella propose un modèle pour les systèmes d'information basés sur une structure hypertexte [Luca90]. Son approche consiste à associer les techniques de recherche d'informations à des mécanismes d'inférence. En recherche d'informations, le système répond aux requêtes de l'utilisateur en localisant les documents qui contiennent l'information demandée. Dans ce type de systèmes, les nœuds du graphe qui organise les documents sont autonomes et les liens qui les relient ne sont utilisés que par les stratégies de recherche d'informations.

Dans un système hypertexte, l'utilisateur utilise les liens pour parcourir le graphe des documents. D. Lucarella organise ce graphe en deux parties pour éviter à l'utilisateur de se "perdre" dans la complexité inévitable des liens qui relient les documents. Pour cela, il associe aux documents un ensemble de concepts qui sont pertinents pour le domaine et qui sont reliés entre eux d'une part, et avec les documents d'autre part. Le système hypertexte est alors organisé en deux couches. La couche des concepts fournit une structure d'index pour accéder aux documents. L'activité de parcours des liens hypertexte est alors distinguée de la recherche d'informations. Le parcours des liens s'effectue à l'intérieur d'une couche, la recherche d'informations consiste à rechercher pour un concept les documents qui le concernent. Le parcours du graphe des concepts constitue ainsi une opération de filtrage qui conditionne la recherche d'informations dans le graphe des documents.

Cette organisation d'un système hypertexte est bien adaptée pour organiser les éléments de la trace et les documents qui les décrivent.

Structure de la trace dans TRAM

Le problème de la trace dans un système à base de connaissances réactif est lié au volume d'informations qu'elle contient. En effet, les éléments qui sont pris en compte dans la modélisation de la résolution doivent figurer dans la trace, de même que leurs effets ou absence d'effet. A ce volume d'éléments de la trace, s'ajoute la complexité des documents qui les décrivent, c'est-à-dire les connaissances explicatives du domaine.

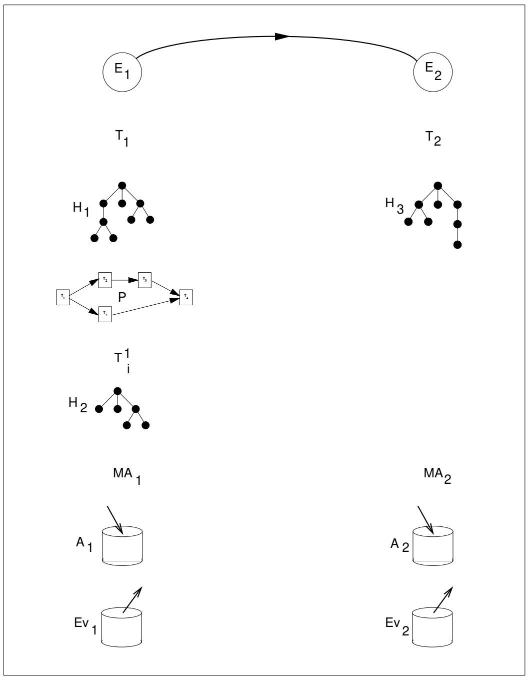

FIG. IV.5 - *Structure hiérarchique de la trace dans le système TRAM*

La trace construite à partir de la résolution dans le système TRAM est structurée de façon hiérarchique. La figure IV.5 présente un exemple de la trace. Plusieurs niveaux sont identifiés. Les états (E_1, E_2) de l'automate qui spécifie le comportement réactif du système identifient chacun un but et la tâche dont ils requièrent la réalisation. A cette tâche correspond la hiérarchie (H_1, H_3) de méthodes qui permettent de la réaliser. Pour une tâche composite, un plan (P) définit sa décomposition en sous-tâches (T_i^1) auxquelles sont associées des hiérarchies (H_2) de méthodes. Après classification de l'instance qui modélise l'état courant de la résolution, une méthode (MA_1) est sélectionnée et exécutée (A_1). Les mises à jour de la base qu'elle réalise provoquent des événements (E_1) qui valident des transitions. La transition valide qui est tirée opère un changement d'état de l'automate, et une nouvelle tâche (T_2) est désignée. Si cette tâche est élémentaire, sa hiérarchie (H_3) de méthodes permet de sélectionner une méthode (MA_2) qui à son tour, met à jour la base.

La structure de la trace produite par TRAM est donc hiérarchique en ce qui concerne les différents niveaux (état, tâche, hiérarchie ...) et séquentielle à l'intérieur de chaque niveau. La séquence des éléments d'un niveau traduit la succession des étapes de la résolution qui les ont générés.

Description des éléments de la trace

A chacun des éléments qui constituent la trace correspondent des connaissances explicatives qui les décrivent. Nous avons supposé que ces connaissances descriptives sont disponibles sous la forme de documents de différentes natures. Pour un élément donné, ses propriétés qui sont décrites constituent la caractérisation de cet élément que le système explicatif peut exploiter. Par exemple, les modules algorithmiques qui constituent les méthodes associées aux tâches élémentaires sont décrits par leur spécification. La carte de description d'un module spécifie son comportement par rapport à la modélisation de la résolution. Elle permet de répondre aux questions qui concernent l'activabilité du module, le programme exécuté et ses arguments, ainsi que les mises à jour de la base que le module effectue. Les configurations de contrôle décrivent les événements qui sont pris en compte, et l'ordre sur les transitions de l'automate justifie les transitions valides qui ont été tirées.

La gestion des possibilités d'activation fournit l'ensemble des données qui rendaient activable une action. Si cette gestion est très coûteuse en volume et n'est donc pas envisageable pour toutes les actions que contrôle le système, elle permet de répondre à une question de type *"Pourquoi pas?"* à partir de la présence ou de l'absence de la possibilité d'activation qui correspond à la question.

L'utilisation d'un système hypertexte a pour objectif d'organiser la trace avec les connaissances explicatives du domaine.

Organisation de l'hypertrace

Nous reprenons la structure des systèmes hypertextes proposés par D. Lucarella. Les éléments de la trace constituent les concepts et les connaissances explicatives qui les décrivent sont regroupées dans le niveau des documents. Le niveau de la trace est organisé par la structure hiérarchique de la trace d'une part, et par les séquences d'éléments d'autre part. L'utilisateur du système hypertexte parcourt les niveaux et liens qui les relient entre eux pour "naviguer" dans la trace.

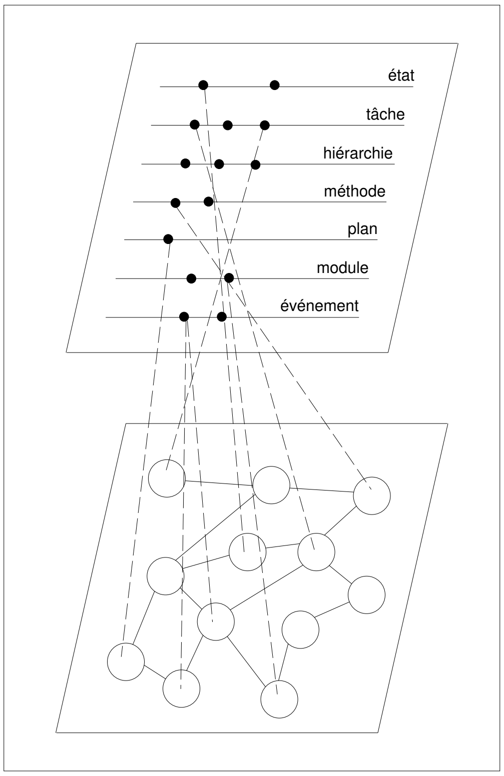

FIG. IV.6 - *Organisation de l'hypertrace*

Les informations qui concernent un élément sont obtenues par la recherche des documents dans la couche inférieure. Cette recherche est effectuée en utilisant les liens qui relient chaque élément de la trace avec les documents qui le décrivent.

Génération de l'hypertrace

La génération de l'hypertrace concerne les liens qui relient les éléments de la trace et les documents. Ces liens ne peuvent être créés *a priori* car ils représentent un volume trop important, pour lequel on ne saurait pas garantir la place mémoire disponible. Aussi les liens sont-ils générés dynamiquement à partir du parcours qu'effectue l'utilisateur dans la couche de la trace. Cette génération dynamique consiste à produire les liens qui relient l'élément consulté avec les documents qui le concernent.

La génération des liens prend en compte le niveau de l'élément examiné, sa position dans le niveau ainsi que les connaissances explicatives actives comme le modèle de l'utilisateur et les stratégies explicatives qui identifient les documents intéressants pour un type d'utilisateur. La génération des liens constitue donc bien une inférence comme le spécifie D. Lucarella.

Evaluation de l'hypertrace pour l'explication

L'hypertrace est mise à la disposition de l'utilisateur qui, en parcourant la trace déclenche les inférences qui génèrent les liens avec la couche des documents. Il s'agit donc bien d'un dialogue et l'utilisateur peut être considéré comme un interlocuteur.

Le principal avantage de ce type d'approche est dû à la construction incrémentale du système. En effet, il est possible de valider cette approche en limitant les niveaux et le nombre d'éléments dans chaque niveau de la trace. La caractérisation de chaque élément peut être étendue progressivement pour intégrer les différents documents qui le concernent.

S'agit-il d'une explication ? Les informations que fournit l'hypertrace constituent une description de l'opération de traitement de l'information pour la résolution. Si l'on se réfère à la définition de l'explication que nous avons retenue, qui est celle proposée par M. Wick (voir IV.4.2), l'hypertrace est donc bien un système explicatif qui permet à l'utilisateur de construire sa propre description de la résolution.

IV.5 Conclusion

L'organisation client-serveur de TRAM définit un système informatique qui constitue une plateforme d'accueil pour des modules algorithmiques externes et dont le rôle consiste à organiser la coopération entre ces modules. Les nombreux problèmes soulevés par le décalage qui existe entre le langage utilisé pour représenter les connaissances et le langage dans lequel sont écrits les modules externes sont abordés au niveau de l'architecture par un protocole de gestion polymorphe des données. La spécification déclarative du comportement des modules algorithmiques par rapport à la modélisation de la résolution permet d'intégrer l'existant pour définir des applications en vraie grandeur.

Comme perspective pour la réalisation, nous envisageons d'utiliser le langages réactif ESTEREL [Berr87] pour la programmation d'une nouvelle version du superviseur. ESTEREL permet en effet d'intégrer les structures de données dans l'automate qu'il définit.

La perspective que nous avons proposée pour l'explication, mais que nous n'avons pas pu mettre en œuvre, est une proposition pour la production de l'explication dans un système à base de connaissances réactif. Elle aboutit à un système d'information de type hypertexte qui est mis à la disposition de l'utilisateur. Cette approche est motivée par le bilan décevant des systèmes explicatifs. Elle se trouve confortée par la nouvelle orientation des travaux sur la production de l'explication qui se consacrent désormais à la production coopérative de l'explication.

Conclusion

Dans ce travail nous avons étudié la problématique de la réactivité dans un $\mathcal{ERP}$. La réactivité intervient dans les systèmes autonomes de résolution de problèmes, mais également dans les systèmes coopératifs : elle permet en particulier de détecter les possibilités d'activation des différents modules algorithmiques disponibles. Ces événements d'activabilité peuvent être proposés à l'utilisateur et donc contribuer à la coopération.

Nous avons proposé une architecture de contrôle réactif sous la forme d'un système de production d'objets qui gère la coopération de modules algorithmiques en résolution de problèmes. Elle est organisée en trois couches : un superviseur qui spécifie le comportement réactif, un gestionnaire de tâches qui assure la réalisation des tâches et de leurs méthodes, et une base d'objets dont l'état courant est caractérisé par les mécanismes de classification d'instances et de détection d'événements.

Problèmes abordés

Les points que nous avons abordés sont les suivants :

La modélisation de la résolution dans un système réactif nécessite la mise en œuvre d'un modèle de données qui permet de caractériser les éléments de la résolution. Pour assurer cette fonction, nous avons proposé un modèle objet qui caractérise les objets par le mécanisme de classification d'instances et qui caractérise l'état courant de la résolution par la détection et la gestion des événements. Pour définir ces événements, nous avons introduit la notion de configuration qui spécifie l'ensemble des tuples d'instances qui satisfont une condition donnée. Le modèle objet est homogène dans la mesure où les configurations définissent de nouvelles classes dont les instances sont gérées par le mécanisme d'évaluation de la base. Afin de prendre en compte les contraintes d'efficacité dans un système réactif, nous avons proposé des algorithmes de calcul incrémental des configurations.

La modélisation des connaissances de contrôle non réactif par les tâches et les méthodes est puissante et compatible avec les plans d'action qui sont utilisés en calcul scientifique. La sélection dynamique de méthodes dans un $\mathcal{ERP}$ réactif a pour but de résoudre les conflits de méthodes. Pour cela, nous avons proposé une approche mixte, qui combine la sélection de méthodes par sélection hiérarchique et la gestion des possibilités d'activation. Cette approche permet également d'utiliser les possibilités d'activation comme des événements qui sont pris en compte dans la couche réactive du contrôle, et donc de mettre en œuvre un mode de contrôle opportuniste.

L'architecture client-serveur apporte des solutions aux problèmes liés à l'intégration de logiciels externes dans un système à base de connaissances. En particulier, la spécification déclarative du comportement des modules algorithmiques permet de limiter la charge de programmation.

Perspectives

La conception des $\mathcal{ERP}$ constitue un objectif dont l'importance est due au poids de l'existant. Par rapport à ce que nous avons proposé, plusieurs perspectives sont envisageables.

Une tâche immédiate concerne la validation de la modélisation du comportement des modules algorithmiques sur des projets qui utilisent de grosses bibliothèques. Plusieurs projets sont en cours, mais aucun n'exploite encore l'utilité telle que nous l'avons définie. La mise en œuvre de notre approche soulève en effet le problème du calcul de l'utilité par rapport aux différents critères de contrôle. L'expression de l'utilité dépend fortement du domaine d'application.

Nous n'avons encore pas étudié les aspects méthodologiques pour la construction des hiérarchies de méthodes associées aux tâches. Or le domaine de l'acquisition des connaissances pour la conception des systèmes à base de connaissances est actif et propose plusieurs méthodes qui peuvent être envisagées.

La rationalité limitée constitue l'une des réponses aux problèmes du temps réel dans les systèmes à base de connaissances. Elle consiste à utiliser la théorie de la décision pour prendre en compte les limitations des ressources du système dans la gestion des connaissances. Nous avons illustré sa mise en application en définissant un mécanisme de sélection rationnelle de méthodes. Il reste à concevoir un mécanisme d'évaluation rationnelle de l'état courant de la base pour que la détection des événements ne soit pas trop longue par rapport à la criticité de certains événements. Ces deux aspects de la rationalité limitée concernent l'utilité de la classification d'instances par rapport à des objectifs identifiés, la sélection de méthodes dans le premier cas et la détection d'événements dans le second. Pour aborder la rationalité limitée de la classification d'instances indépendamment de l'objectif, il est nécessaire de concevoir un système rationnel de représentation des connaissances qui offre les services nécessaires à la mise en œuvre de la théorie de la décision.

Références

[Abit91] S. Abiteboul et E. Simon. Fundamental properties of deterministic and non deterministic extensions of datalog. *Theoretical Computer Science*, 78(1):137–158, 1991.

[Abit92] S. Abiteboul et A. Van Gelder. Optimizing active databases using the split technique. Actes J. Biskup et R. Hull, éditeur, *International Conference on Database Theory*, pages 171–187. Springer-Verlag, octobre 1992.

[Abu90] S. Abu-Hakima et F. Oppacher. Improving explanations in knowledge-based systems: RATIONALE. *Knowledge Acquisition*, 2(4):301–343, 1990.

[Alle84] J.F. Allen. Towards a general theory of actions and time. *Artificial Intelligence*, 23(1):123–154, 1984.

[Ash93] D. Ash, G. Gold, A. Seiver et B. Hayes-Roth. Guaranteeing real-time response with limited-resources. *Artificial Intelligence in Medicine*, 5(1):49–66, 1993.

[Baad91] F. Baader et P. Hanschke. A scheme for integrating concrete domains into concept languages. Actes *IJCAI*, pages 452–457, Sydney (Aus), août 1991.

[Bach91] B. Bachimont. A logical framework to manage coherence and convergence in a blackboard: a proposal. Actes *Workshop on Blackboard Systems*, Anaheim (CA US), 1991. AAAI.

[Bach92] B. Bachimont. *Le contrôle dans les systèmes à base de connaissances*. Hermes, 1992.

[Bala90] N. Balacheff. Problèmes de la production d'une explication : aspects conceptuels et langagiers. *Revue d'Intelligence Artificielle*, 4(2):149–160, 1990.

[Bals91] H. Balsters et C.C de Vreeze. A semantics of object-oriented sets. Actes *International Workshop on Database Programming Languages: Bulk Types and Persistent data*, pages 203–217, Nafplion (Gr), août 1991. Morgan Kaufman.

[Bapt92] P. Baptiste et D. Chappe. Intérêt des réseaux de Petri à terminaison propre pour le pilotage temps réel d'ateliers flexibles. *Automatique, Productique et Informatique Industrielle*, 26:423–435, 1992.

[Bara90] P. Baras, J. Blum, J.C. Paumier, F. Rechenmann et P. Witomski. EVE : an object-centered knowledge-based PDE solver. Actes *2nd International Conference on Expert System for Numerical Computing*, pages 24–26, West-Lafayette (IN US), avril 1990.

[Bara92] F. Barachini, H. Mistelberger et A. Gupta. Run-time prediction for production systems. Actes *10th AAAI Conference*, pages 478–485, San Jose (CA US), juillet 1992.

[Berr87] G. Berry and P. Courroné and G. Gonthier. Programmation synchrone des systèmes réactifs : le langage ESTEREL. *Technique et Sciences Informatique*, 6(4):305–316, 1987.

[Bert92] E. Bertino, M. Negri, G. Pelagatti et L. Sbatella. Object-oriented query languages: The notion and the issues. *IEEE Transactions on Knowledge and Data Engineering*, 4(3):223–237, 1992.

[Borg89a] A. Borgida. Type systems for querying class hierarchies with non-strict inheritance. Actes *8th ACM Symposium on Principles of Database Systems*, pages 394–400, mars 1989.

[Borg89b] A. Borgida, R.J. Brachman, D.L. McGuiness et L.A. Resnick. Classic: a structural data model for objects. Actes *ACM SIGMOD International Conference on Management of Data*, pages 59–67, Portland (OR US), 1989.

[Brac85] R.J. Brachman et J. Schmolze. An overview of the KL-ONE knowledge representation system. *Cognitive Science*, 9(2):171–216, 1985.

[Breu85] J. Breuker et B. Wielenga. KADS: Structured knowledge acquisition for expert systems. Actes *5th International Workshop on Expert Systems*, Avignon (Fr), 1985.

[Broo91] R.A. Brooks. Intelligence without representation. *Artificial Intelligence*, 47(1):139–159, 1991.

[Capp93] C. Capponi et M. Chaillot. Construction incrémentale d'une base de classes correcte du point de vue des types. Actes *Journées Apprentissage, Validation et Acquisition*, St Raphaël (Fr), mars 1993.

[Card85] L. Cardelli. Typechecking dependent types and subtypes. Actes *Workshop on Foundations of Logic and Functional Programming*, pages 45–57, 1985.

[Card91] L. Cardelli et J.C. Mitchell. Operations on records. *Mathematical Structures in Computer Science*, 1(1):3–48, mars 1991.

[Carv92] N. Carver et V. Lesser. The evolution of blackboard control architectures. Rapport CMPSCI-92-71, University of Massachusetts, Amherst, octobre 1992.

[Ceri89] S. Ceri, G. Gottlob et L. Tanca. What you always wanted to know about Datalog and never dared to ask. *IEEE Transactions on Knowledge and Data Engineering*, 1(1):146–166, 1989.

[Chai89a] M. Chaillot. Insertion d'une nouvelle classe dans une hiérarchie de classes : une approche par les types. Rapport de DEA, INPG-UJF, Grenoble (Fr), septembre 1989.

[Chai89b] M. Chaillot et P. Girard. EVE: un environnement de résolution de problèmes pour les équations aux dérivées partielles. Rapport de fin d'études, ENSIMAG, Grenoble (Fr), juin 1989.

[Chai92a] M. Chaillot. Une approche client/serveur pour la conception d'un système blackboard. Actes *Rencontre Nationale des Jeunes Chercheurs en Intelligence Artificielle*, pages 122–131, Rennes (Fr), septembre 1992.

[Chai92b] M. Chaillot et E. Crochon. Declarative specifications of blackboard updates. Actes *ERCIM Workshop on Theoretical and Experimental Aspects of Knowledge Representation*, pages 199–204, Pisa (It), May 1992.

[Chai93] M. Chaillot et E. Crochon. Classification rationnelle d'instances pour la sélection dynamique de méthodes. Actes *Représentation Par Objets*, pages 225–234, La Grande Motte (Fr), juin 1993. EC2.

[Chai94] M. Chaillot et E. Crochon. Environnements réactifs de résolution de problèmes. Actes *Reconnaissance des Formes et Intelligence Artificielle*, pages 293–303, Paris (Fr), janvier 1994.

[Chan86] B. Chandrasekaran. Generic tasks in knowledge-based reasoning : High-level building blocks for expert system design. *IEEE Expert*, Fall:23–30, 1986.

[Chan92] B. Chandrasekaran, T.R. Johnson et J.W. Smith. Task-structure analysis for knowledge modelling. *Communication of the ACM*, 35(9):124–137, septembre 1992.

[Chen91] J.P. Cheney et Y.N. Huang. Set-oriented propagation of updates into transitively closed relations. Actes *International Conference on Deductive and Object-Oriented Databases*, Munich (A), décembre 1991. Springer-Verlag.

[Ches89] D. Chessel et S. Dolédec. ADE software: Multivariate analyses, graphical display for environmental data, version 3.1. User's manual, URA CNRS 1451, Université Claude Bernard, Lyon I, octobre 1989.

[Chev91] F. Chevenet et J. Willamowski. Vers la modélisation des connaissances méthodologiques en analyse des données. Actes *EDORA*, Lyon (Fr), mai 1991.

[Chev93] F. Chevenet, F. Jean-Marie et J. Willamowski. SLOT : a cooperative problem-solving environment in exploratory data analysis. Actes *49 th Session of International Institute*, pages 255–256, Florence, août 1993.

[Clan81] W.J. Clancey et R. Letsinger. Neomycin: Reconfiguring a rule-based expert system for application to teaching. Actes *IJCAI*, Vancouver (Can), août 1981.

[Clan85] W. Clancey. Heuristic classification. *Artificial Intelligence*, 27(1):289–350, 1985.

[Clan92] W. Clancey. Model construction operators. *Artificial Intelligence*, 53(1):1–115, janvier 1992.

[Clem93] V. Clement et M. Thonnat. A knowledge-based approach to integration of image processing procedures. *CVGIP : Image Understanding*, 57(2):166–184, 1993.

[Croc93] E. Crochon, F. Vacherand, M. Chaillot et M. Antonakios. Hierarchical control system with high reactivity for autonomous mobile robot. Actes *IFAC Workshop on Intelligent Autonomous Vehicles*, pages 421–426, Southampton (UK), avril 1993.

[Curr91] K. Currie et A. Tate. O-Plan : the open planning architecture. *Artificial Intelligence*, 52(1):49–86, novembre 1991.

[Davi80] R. Davis. Meta-rules: reasoning about control. *Artificial Intelligence*, 15(3):177–222, 1980.

[Davi90] J-M. David et J-P. Krivine. Structuration du raisonnement et explications. *Revue d'Intelligence Artificielle*, 4(2):77–88, 1990.

[Dean91] T. Dean. Decision-theoretic control of inference for time-critical applications. *International Journal of Intelligent Systems*, 6(4):417–441, juillet 1991.

[Deck90] K.S. Decker, V.R. Lesser et R.C. Whitehair. Extending a blackboard architecture for approximate processing. *Journal of Real-Time Systems*, 2(1):47–79, 1990.

[Dien90] R. Dieng. Méthodes et outils d'acquisition des connaissances. Rapport RR-1319, INRIA, Sophia-Antipolis, juillet 1990.

[Doyl88] J. Doyle. Knowledge representation and rational self-government. Actes *2nd Int. Conf. on Theoretical Aspects of Reasoning about Knowledge*, pages 345–354, Pacific Cirove (CA US), 1988.

[Doyl89] J. Doyle. Rational control of reasoning in artificial intelligence. Actes *The Logic of Theory Change, L.N.C.S. 465*, pages 19–48, Konstanz (A), octobre 1989.

[Doyl90] J. Doyle. Rationality and its roles in reasoning. Actes *8th AAAI Conference*, pages 1093–1100, Boston (MA US), 1990.

[Doyl91] J. Doyle et R.S. Patil. Two theses of knowledge representation: language restrictions, taxonomic classification, and the utility of representation services. *Artificial Intelligence*, 48(3):261–297, avril 1991.

[Dzie90] D. Dzierzgowski. Quatre exemples de langages ou environnements pour le développement de programmes où le temps intervient. *Technique et science informatique*, 9(4):289–312, 1990.

[Ermi93] J.-L. Ermine. *Génie logiciel et Génie cognitif*, volume 1,2 of *TEC DOC*. Lavoisier, 1993.

[Esti93] T. Estier, G. Falquet et M. Léonard. F2: an evolution-oriented database system. Rapport CUI-TR-69, Centre Universitaire d'Informatique, University of Geneva (CH), 1993.

[Etzi89] O. Etzioni. Tractable decision-analytic control. Actes *1st Conference on Principles of Knowledge Representation and Reasoning*, pages 114–125, Los Altos (CA US), 1989.

[Etzi91] O. Etzioni. Embedding decision-analytic control in a learning architecture. *Artificial Intelligence*, 49:129–159, 1991.

[Euze91] J. Euzenat et L. Maesano. An architecture for selective forgetting. Actes *8th Conference on Artificial Intelligence and Simulation of Behaviour*, London (UK), 1991.

[Euze93a] J. Euzenat. Définition abstraite de la classification et son application aux taxonomies d'objets. Actes *Représentation par objets*, La Grande Motte (Fr), juin 1993. EC2.

[Euze93b] J. Euzenat. On a purely taxonomic and descriptive meaning for classes. Actes *IJCAI Workshop on Object Based Representation Systems*, Chambery (Fr), août 1993.

[Fabr92] F. Fabret, M. Régnier et E. Simon. Optimizing incremental computation of Datalog programs with non-deterministic semantics. Actes *International Conference on Database Theory*, pages 155–170, Berlin (Al), octobre 1992. LNCS 646, Springer-Verlag.

[Fage92] F. Fages et R. Lissajoux. Sémantique opérationnelle et compilation des systèmes de production. *Revue d'Intelligence Artificielle*, 6(4):431–456, 1992.

[Falq90] G. Falquet. F2 : an object-oriented database model with semantic contexts. Rapport CUI-TR-52, Centre Universitaire d'Informatique, University of Geneva (CH), 1990.

[Ferg92] I.A. Ferguson. *Touring Machines : an architecture for dynamic, rational, mobile agents.* Doctorat, University of Cambridge (UK), 1992.

[Forg81] C.L. Forgy. OPS-5 user's manual. Rapport CMU-CS-81-135, Carnegie Mellon University, Pittsburgh (PA), juillet 1981.

[Freu90] Y. Freundlich. Knowledge bases and databases. *Computer*, 23(11):51–57, 1990.

[Gall91] E. Gallopoulos, E. Houstis et J.R. Rice, éditeur. *Future research directions in problem solving environments for computational science*, Washington (DC), avril 1991.

[Gana91] J-G. Ganascia. L'hypothèse du knowledge level: théorie et pratique. Actes *Les sciences cognitives en débat*, pages 57–71. CNRS, 1991.

[Gatz93] S. Gatziu et K.R. Dittrich. Events in an active object-oriented database system. Rapport 93.11, Institut für Informatik der Universität Zürich (CH), avril 1993.

[Geor89] M.P. Georgeff et F.F. Ingrand. Decision-making in an embedded reasoning system. Actes *IJCAI*, pages 972–978, Detroit (MI US), 1989.

[Ghal88] M. Ghallab. Compilation de bases de connaissances. Actes *Journées Nationales du PRC-IA*, pages 231–253, Toulouse (Fr), mars 1988. Teknea.

[Goel91] A.K. Goel. Knowledge compilation: a symposium. *IEEE Expert*, pages 71–93, avril 1991.

[Grou88] Groupe de réflexion temps réel. Le temps réel. Rapport, CNRS, 1988.

[Gué93] S. Guédon, B. Flament et R. Blanpain. Système multiexpert pour la surveillance océanographique à partir d'un réseau de magnétomètres haute sensibilité. Actes *GRETSI*, à paraitre 1993.

[Hami92] B. Hamidzadeh et S. Shekhar. Can real-time search algorithms meet deadlines? Actes *10th AAAI Conference*, pages 486–491, San Jose (CA US), juillet 1992.

[Hans92] E.N. Hanson et J. Widom. An overview of production rules in database systems. Rapport CIS-TR-92-031, University of Florida, Gainesville (FL US), septembre 1992.

[Haye85] B. Hayes-Roth. A blackboard architecture for control. *Artificial Intelligence*, 26(2):251–321, 1985.

[Haye90] B. Hayes-Roth. Architectural foundations for real-time performance in intelligent agents. *The Journal of Real-Time Systems*, 2(1-2):99–125, 1990.

[Horv89] E.J. Horvitz. Rational metareasoning and compilation for optimizing decisions under bounded resources. Actes *2nd International Conference on Computational Intelligence*, pages 205–216, 1989.

[Hous93] E.N. Houstis, J.R. Rice et R. Vichnevetsky, éditeur. *3th International Conference on Expert Systems for Numerical Computing*, Purdue (IN US), mai 1993.

[Howe90] A.E. Howe et P.R. Cohen. Responding to environment change. Actes *DARPA Workshop on Innovative Approaches to Planning Scheduling and Control*, pages 85–92, San Diego (Ca US), novembre 1990. Phoenix.

[Hurs93] A.R. Hurson, S.H. Pakzad et J. Cheng. Object-oriented database management systems: Evolution and performance issues. *Computer*, pages 48–58, février 1993.

[Ingr92] F.F. Ingrand, M.P. Georgeff et A.S. Rao. An architecture for real-time reasoning and system control. *IEEE Expert*, pages 34–44, décembre 1992.

[Jeus92] M. Jeusfeld et M. Staudt. Query optimization in deductive object bases. Actes *Query processing in object-oriented, complex-object, and nested relation databases*. Morgan Kaufmann, 1992.

[Karb90] W. Karbach, M. Linster et A. Voss. Models, methods, roles and tasks : many labels - one idea ? *Knowledge Acquisition*, 2(4):279–299, décembre 1990.

[Kim90] W. Kim. Object-oriented databases : Definition and research directions. *IEEE Transactions on Knowledge and Data Engineering*, 2(3):327–341, septembre 1990.

[Klin91] G. Klinker, C. Bhola, G. Dallemagne, D. Marques et J. McDermott. Usable and reusable programming constructs. *Knowledge Acquisition*, 3(2):117–135, juin 1991.

[Koen89] A. Koenig. *TRAM : une architecture blackboard dédiée à la robotique autonome mobile*. Doctorat, INPG, Grenoble (Fr), juin 1989.

[Kuma92] V. Kumar. Algorithms for constraint satisfaction problems: A survey. *AI Magazine*, 13(1):32–44, 1992.

[Lair87] J.E. Laird, A. Newell et P.S. Rosenbloom. SOAR: an architecture for general intelligence. *Artificial Intelligence*, 33(1):1–64, 1987.

[Lark90] J.S. Lark, L.D. Erman, S. Forrest, K.P. Gostelow, F. Hayes-Roth et D.M. Smith. Concepts, methods, and languages for building timely intelligent systems. *The Journal of Real-Time Systems*, 2(1):127–148, 1990.

[Lema92] B. Lemaire. Hypothetical reasoning within the blackboard model for constructing explanations. Actes *ECAI*, pages 244–248, Vienne (Au), août 1992.

[Leve87] H.J. Levesque et R.J. Brachman. Expressiveness and tractability in knowledge representation and reasoning. *Computational Intelligence*, 3(1):78–93, 1987.

[Luca90] D. Lucarella. A model for hypertext-based information retrieval. Actes INRIA, éditeur, *Hypertext: Concepts, Systems, and Applications*, pages 81–94. European Conference on Hypertext, Cambridge University Press, novembre 1990.

[Maes90] P. Maes. Situated agents can have goals. *Robotics and Autonomous Systems*, 6(1,2):49–70, 1990.

[Mari90] O. Mariño, F. Rechenmann et P. Uvietta. Multiple perspectives and classification mechanism in object-oriented representation. Actes *9th ECAI*, pages 425–430, Stockholm (SW), August 1990.

[Mari93] O. Mariño. *Classification dans les systèmes de représentation par objets.* Doctorat, Université Joseph Fourrier, Grenoble, à paraître 1993.

[Marr82] D. Marr. *Vision.* W.H. Freeman, 1982.

[McCa69] J. McCarthy et P.J. Hayes. Some philosophical problems from the standpoint of artificial intelligence. *Machine Intelligence*, 4:463–502, 1969.

[McDe82] D. McDermott. A temporal logic for reasoning about processes and plans. *Cognitive Science*, 6:101–155, 1982.

[McDe88] J. McDermott. *Automating Knowledge Acquisition for Expert Systems*, chapter Preliminary steps toward a taxonomy of problem-solving methods, pages 225–256. S. Marcus, Kluwer Academic edition, 1988.

[Méd93] C. Médigue, J. Willamowski, O. Schmeltzer, P. Uvietta, F. Rechenmann, F. Chevenet, G.Perrière and C. Gautier. Modeling tasks for problem solving in molecular biology. Actes *Workshop Artificial Intelligence and Genome*, pages 67–76, Chambery, novembre 1993.

[Mish92] P. Mishra et M.H. Eich. Join processing in relationnal databases. *ACM Computing Surveys*, 24(1):63–113, mars 1992.

[Moor88] J.D. Moore et W.R. Swartout. Explanation in expert systems: a survey. Rapport ISI/RR-88-228, Information Science Institute, University of Southern California, décembre 1988.

[Napo92] A. Napoli. Représentation à objets et raisonnement par classification en intelligence artificielle. Thèse d'état, Université de Nancy (Fr), 1992.

[Nebe90] B. Nebel. Terminological reasoning is inherently intractable. *Artificial Intelligence*, 43(2):235–249, 1990.

[Newe82] A. Newell. The knowledge level. *Artificial Intelligence*, 18(1):87–127, 1982.

[Newe93] A. Newell et D. Steier. Intelligent control of external software systems. *Artificial Intelligence in Engineering*, 8(1):3–21, 1993.

[Pfal88] J.L. Pfaltz. Implementing set operators over class hierarchies. Rapport IPC-TR-88-04, School of Engineering, University of Virginia, août 1988.

[Pfal92] J.L. Pfaltz et J.C. French. Multiple inheritance and the closure of set operators in class hierarchies. Rapport IPC-TR-92-04, School of Engineering, University of Virginia, juin 1992.

[Piqu92] J-D. Piques. *Systèmes experts en contexte opérationnel: optimisation des performances et traitement de connaissances incertaines.* Doctorat, INPG, Grenoble (Fr), janvier 1992.

[Poll92] M.E. Pollack. The uses of plans. *Artificial Intelligence*, 57(1):43–68, 1992.

[Puer93] A.R. Puerta, S.W. Tu et M.A. Musen. Modelling tasks with mechanisms. *International Journal of Intelligent Systems*, 8(1):129–152, 1993.

[Qian91] X. Qian et G. Wiederhold. Incremental recomputation of active relationnal expressions. *IEEE Transactions on Knowledge and Data Engineering*, 3(3):337–341, 1991.

[Rech85] F. Rechenmann. SHIRKA : Mécanismes d'inférence sur une base de connaissances centrée-objet. Actes *RFIA*, pages 1243–1254, Grenoble (Fr), novembre 1985.

[Rech91] F. Rechenmann. Modelling tasks and methods in a knowledge-based pde solver. Actes *IMACS*, Dublin (Ir), juin 1991.

[Rech92] F. Rechenmann et B. Rousseau. A development shell for knowledge-based systems in scientific computing. Actes E.N. Houstis et J.R. Rice, éditeur, *Expert Systems for Numerical Computing.* Elsevier Science Publishers, 1992.

[Regn90] P. Regnier. Planification : historique, principes, problèmes et méthodes (de GPS à ABTWEAK). Rapport IRIT-90, IRIT, Toulouse (Fr), juin 1990.

[Rous88] B. Rousseau. *Vers un environnement de résolution de problèmes en biométrie. Apport des techniques de l'intelligence artificielle et de l'interaction graphique.* Doctorat, Univ. Claude Bernard, Lyon I France, 1988.

[Rund92] E.A. Rundensteiner et L. Bic. Set operations in object-based data models. *IEEE Transactions on Knowledge and Data Engineering*, 4(3):382–398, juin 1992.

[Russ91] S. Russell et E. Wefald. Principles of metareasoning. *Artificial Intelligence*, 49:361–395, 1991.

[Schw92] U.M. Schwuttke et L. Gasser. Real-time metareasoning with dynamic trade-off evaluation. Actes *10th AAAI Conference*, San Jose (CA US), juillet 1992.

[Shor74] E.H. Shortliffe. *MYCIN: a rule-based computer program for advising physician regarding antimicrobial therapy selection.* Doctorat, Stanford University (CA US), 1974.

[Sinh92] A. Sinha. Client-server computing. *Communications of the ACM*, 35(7):77–98, 1992.

[Smit93] J.W. Smith et A. Bayazitoglu. Exploring the relationship between rationality and bounded rationality in medical knowledge-based systems. *Artificial Intelligence in Medicine*, 5(2):125–142, 1993.

[Stra90] D.D. Straube. Queries and query processing in object-oriented database systems. Rapport TR90-33, University of Alberta, Edmonton (Can), décembre 1990.

[Stre91] M. Strecker. Formalisation de concepts d'oubli dans les systèmes de maintien du raisonnement. Actes *RFIA*, pages 1281–1292, Lyon (Fr), 1991.

[Swar83] W.R. Swartout. XPLAIN: a system for creating and explaining expert consulting programs. *Artificial Intelligence*, 21(3):285–325, 1983.

[Swar91] W.R. Swartout, C. Paris et J. Moore. Design for explainable expert systems. *IEEE Expert*, 6(3):59–64, 1991.

[Tann91] M.C. Tanner et A.M. Keuneke. The roles of the task structure and domain functional models. *IEEE Expert*, 6(3):50–57, juin 1991.

[Thon93] M. Thonnat, V. Clement et J. Van den Elst. Supervision of perception tasks for autonmous systems: the OCAPI approach. Rapport RR-2000, INRIA, juin 1993.

[Tu89] S.W. Tu, M.G. Kahn, M.A. Musen, J.C. Ferguson, E.H. Shortliffe et L.M. Fagan. Episodic skeletal-plan refinement based on temporal data. *Communications of the ACM*, 32(12):1439–1455, décembre 1989.

[Tyrr93] T. Tyrrell. *Computational Mechanisms for Action Selection.* Doctorat, University of Edinburgh, UK, 1993.

[Wash89] R. Washington et B. Hayes-Roth. Input data management in real-time AI systems. Actes *IJCAI*, pages 250–255, 1989.

[Wick92] M.R. Wick et W.B. Thompson. Reconstructive expert system explanation. *Artificial Intelligence*, 54(1):33–70, 1992.

[Wiel92] B. Wielenga, A.T. Schreiber et J.Breuker. KADS: a modelling approach to knowledge engineering. *Knowledge Acquisition*, 4(1):5–53, 1992.

[Will92] C. Williams, J. Rasure et C. Hansen. The state of the art of visual languages for visualization. Actes A.E.Kaufman et G.M. Nielson, éditeur, *Visualization*, pages 202–209, Boston (MA US), octobre 1992. IEEE Computer Society Press.

[Will93] J. Willamowski. *Coopération dans les environnements de résolution de problèmes : SCARP*. Doctorat, Université Joseph Fourrier, Grenoble (Fr), avril 1993.

[Wood91] W.A. Woods. *Understanding Subsumption and Taxonomy: a Framework for Progress*, chapter 1. Morgan Kaufman, 1991.

[Yen91] J. Yen, R. Neches et R. MacGregor. CLASP: Integrating term subsumption systems and production systems. *IEEE Transactions on Knowledge and Data Engineering*, 3(1):25–32, 1991.