AVERTISSEMENT

Ce document est le fruit d'un long travail approuvé par le jury de soutenance et mis à disposition de l'ensemble de la communauté universitaire élargie.

Il est soumis à la propriété intellectuelle de l'auteur. Ceci implique une obligation de citation et de référencement lors de l'utilisation de ce document.

D'autre part, toute contrefaçon, plagiat, reproduction illicite encourt une poursuite pénale.

Contact : ddoc-theses-contact@univ-lorraine.fr

LIENS

Code de la Propriété Intellectuelle. articles L 122. 4
Code de la Propriété Intellectuelle. articles L 335.2- L 335.10
http://www.cfcopies.com/V2/leg/leg_droi.php
http://www.culture.gouv.fr/culture/infos-pratiques/droits/protection.htm

9/1625

SCN 99/280 B

Département de formation doctorale en informatique
UFR STMIA

Ecole doctorale IAE+M

Environnement d'aide au développement & Bibliothèque de programmes « intelligente »

S.C.D. - U.H.P. NANCY 1 SCIENCES BIBLIOTHÈQUE

THÈSE

présentée et soutenue publiquement le 20 Décembre 1999

pour l'obtention du

Doctorat de l'Université Henri Poincaré - Nancy 1
(spécialité informatique)

par

Philippe HAÏK

Composition du jury

Rapporteurs : Claude GODART, Professeur, Directeur de Recherche à l'INRIA
Sabine MOISAN, Chargé de Recherche à l'INRIA
Karl SCHLECHTA, Professeur à l'Université de Provence

Examinateur : Guy D'URSO, Ingénieur de Recherche à la Division Recherche et Développement d'EDF

Directeurs : François CHARPILLET, Chargé de Recherche à l'INRIA
Jean-Paul HATON, Professeur à l'Institut de France et à l'Université Henri POINCARÉ, Nancy

Laboratoire Lorrain de Recherche en Informatique et ses Applications - UMR 7503

BIBLIOTHEQUE SCIENCES NANCY 1
D 095 130127 4

Loria

Université Henri Poincaré, Nancy I

Résumé

ENVIRONNEMENT D'AIDE AU DEVELOPPEMENT & BIBLIOTHEQUE DE PROGRAMMES « INTELLIGENTE »

par Philippe HAÏK

Directeurs de thèse : Jean-Paul Haton, Professeur, Institut de France et Université Henri Poincaré, Nancy I
François Charpillet, Chargé de Recherche, Inria

Le travail réalisé dans le cadre de cette thèse a consisté à définir et à tester en partie un ***Environnement d'Aide au Développement*** (sorte d'***Environnement de Résolution de Problèmes Générique***) basé sur une ***Bibliothèque de Programmes Intelligente*** (ou ***Programmathèque***) et sur l'utilisation de techniques de ***Pilotage de Programmes*** permettant non seulement de choisir les algorithmes à combiner afin de réaliser la tâche complexe que l'on souhaite automatiser, mais aussi de contrôler leur exécution de sorte à pouvoir éventuellement remettre en question certains choix en fonction des résultats produits observés.

Abstract

COOPERATIVE SOFTWARE DESIGN TOOL & « INTELLIGENT » SOFTWARE LIBRARY

by Philippe HAÏK

Thesis Directors : Jean-Paul Haton, Professor, Institut de France et Université Henri Poincaré, Nancy I
François Charpillet, Chargé de Recherche, Inria

The realised work consisted in defining and testing a ***Co-operative Software Design Tool*** (a kind of ***Generic Problem Solver***) based on an ***« Intelligent » Software Library*** and on the use of ***Program Supervision Techniques***. The aim was to provide the user with a tool that can select autonomously (and almost automatically) and control the basic algorithms that must be combined in order to achieve the intended task. The already produced results and the shifts in the environment are dynamically taken into account during the selection and the control processes.

S.C.D. - U.H.P.
NANCY 1
SCIENCES
BIBLIOTHÈQUE

A celle en qui je puise courage et inspiration et auprès de la quelle je trouve réconfort depuis plus de 12 ans…

mon épouse, Roxane

A la mémoire de mon grand-père : j'espère me montrer digne de son parcours exceptionnel qui constitue pour moi un exemple et qui continue de me guider dans la voie professionnelle que j'ai choisie.

« Savoir que l'on sait ce que l'on sait, et savoir que l'on ne sait pas ce que l'on ne sait pas : voilà la véritable intelligence ». (Confucius)

S.C.D. - U.H.P.
NANCY 1
SCIENCES
BIBLIOTHÈQUE

TABLE DES MATIÈRES

S.C.D. - U.H.P.
NANCY 1
SCIENCES
BIBLIOTHÈQUE

S.C.D. - U.H.P. NANCY 1 SCIENCES BIBLIOTHÈQUE

TABLE DES ILLUSTRATIONS

S.C.D. - U.H.P.
NANCY 1
SCIENCES
BIBLIOTHÈQUE

REMERCIEMENTS

Je tiens à remercier tous les membres du jury :

- Claude GODART, professeur, directeur de recherche à l'INRIA, qui m'a fait l'honneur de présider ce jury. Qu'il trouve ici l'expression de ma gratitude pour le bienveillant intérêt qu'il a porté à mes recherche et l'appui amical qu'il m'a toujours manifesté.

- Sabine MOISAN, chargée de recherche à l'INRIA, qui a accepté d'être le rapporteur de ce travail. Son regard avisé sur mon travail et ses nombreux conseils m'ont permis d'avancer dans mes recherches et d'augmenter la clarté de ce manuscrit.

- Guy D'URSO, ingénieur de recherche à la Division Recherche & Développement d'Electricité de France, qui a accepté d'encadrer et de juger mon travail, pour l'intérêt qu'il lui a manifesté, pour ses nombreuses remarques qui ont contribué à l'amélioration du présent manuscrit et pour m'avoir ouvert de nouvelles perspectives.

- et Karl SCHLECHTA, professeur à l'Université de Provence, qui a accepté de participer à mon jury de thèse, pour l'intérêt qu'il a porté à mon travail et pour sa disponibilité.

Je tiens aussi à exprimer ma reconnaissance à Jean-Paul HATON qui m'a fait confiance en me proposant ce sujet de recherche et m'a accueilli au sein de ce qui fut l'équipe Reconnaissance de Formes et Intelligence Artificielle.

Un grand merci aussi à François CHARPILLET, mon directeur de thèse. Sa disponibilité, ses qualités humaines et scientifiques, ainsi que sa capacité déjà célèbre (**[Gallone 97]**) à travailler sous contraintes de ressources – et plus spécialement sous contraintes de temps – resteront pour moi des exemples.

Ma gratitude va également à tous les membres du laboratoire qui contribuent, chacun à leur manière, à la bonne ambiance qui y règne.

Merci aux membres de l'Axe RFIA et de l'équipe MAIA pour leur disponibilité, leur gentillesse et leurs conseils. Je remercie tout particulièrement Jean-Michel GALLONE pour

son accueil, ses remarques pertinentes, son humour légendaire et son aide qu'il n'a jamais hésité à me prodiguer.

Merci aussi à Martine Kuhlmann pour sa grande gentillesse, son efficacité et sa générosité.

Je tiens aussi à remercier toute l'équipe du Groupe Traitements Avancés de l'Information, du Département Surveillance, Diagnostique, Maintenance de la Division Recherche & Développement d'Electricité de France, pour leur accueil et leur disponibilité à l'occasion de mes différents séjours sur leur site de Chatou ; je remercie tout particulièrement Benoit RICARD pour son aide, ses conseils, sa grande disponibilité et pour m'avoir fait partager ses connaissances gastronomiques. Merci aussi à tous ceux qui m'ont accueilli depuis ces derniers mois comme un membre à part entière du groupe P21 et qui m'ont aidé et soutenu pendant la rédaction du présent manuscrit et pendant la préparation de mon exposé de soutenance.

Enfin, je tiens à remercier ceux sans qui ce travail n'aurait pu aboutir, ma famille et mes amis. Je remercie tout d'abord mon épouse Roxane, re-lectrice dévouée, mais malchanceuse - parce que contrainte - de la quasi-totalité de mes « productions littéraires », pour son immense dévotion et son soutien de tous les instants ; Je remercie aussi mes parents pour leurs encouragements, leur aide et la confiance qu'ils ont toujours eue en moi, mes beaux-parents pour leur joie de vivre et leurs bonnes recettes de cuisine, Laurette pour sa douceur, Nathalie pour ses TR (et malgré ses fréquents oublis), Isabelle pour ses mails quotidiens, Stéphane pour ses « bons tuyaux », Denise pour ses repas en famille, David, Norbert, Manu, Jean-Emile et les autres courageux du CNK pour tous les bons moments passés ensemble.

LEXIQUE

Base de Connaissances : une base de connaissances est un ensemble d'informations permettant de représenter ce qui est connu sur un domaine particulier. La représentation des connaissances consiste donc à modéliser les différents éléments du domaine, leurs comportements propres et les relations qui existent entre eux.

Bibliothèque de Programmes : une bibliothèque de programmes est un ensemble de programmes fournis, le plus souvent clef en main, dédié à une classe de problème ; les bibliothèques de programmes sont apparues dans les années 60 pour encourager la réutilisation de modules logiciels de grande qualité afin de permettre le transfert du savoir-faire des chercheurs en analyse numérique et en algorithmique vers les utilisateurs industriels de tels algorithmes.

Blackboard System : voir système à tableau noir.

Décomposition de Tâche : la décomposition de tâches consiste à identifier les différentes sous-tâches qui composent une tâche complexe (ou composée). Le but de la décomposition est de réduire progressivement la complexité des tâches à effectuer en isolant des sous-tâches plus simples à résoudre.

DRD : Division Recherche & Développement d'Electricité de France, anciennement Direction des Etudes et Recherches (**DER**).

DSP : Densité Spectrale de Puissance.

Environnement d'Aide au Développement : un environnement d'aide au développement est un système informatique qui fournit à ses utilisateurs tous les outils logiciels nécessaires à la conception, à la programmation, au test et à la validation de nouveaux logiciels.

Environnement de Résolution de Problèmes (ERP) : un environnement de résolution de problème est un système informatique qui fournit à ses utilisateurs toutes les ressources computationnelles nécessaires à la résolution d'une classe de problème ; cela inclut donc notamment la fourniture de méthodes solutions, la sélection automatique ou semi-automatique de méthodes, la possibilité d'ajouter facilement de nouvelles solutions, le choix des ressources matérielles à utiliser ou encore le contrôle des exécutions.

Un tel environnement de résolution de problèmes est qualifié de **coopératifs** (on parle alors d'**ERPC**) s'il peut fonctionner en interaction avec son utilisateur, exploitant ainsi la complémentarité des compétences du système et de l'utilisateur lors de la résolution d'un problème. En effet, un ERPC permet à l'utilisateur d'intervenir à tout moment dans le processus de résolution d'un problème que ce soit pour orienter la résolution par le choix d'une méthode jugée plus adaptée par celui-ci, ou pour valider une étape intermédiaire de la résolution.

Il est qualifié de **distribué** (on parle alors d'**ERPD**) si les ressources nécessaires à la résolution (méthodes, connaissances permettant le contrôle de la résolution, matériel, etc.) sont distribuées.

Enfin il est qualifié de **générique** (on parle alors d'**ERPG**) s'il peut s'appliquer indifféremment à un grand nombre de classes de problèmes, sans être dédié plus particulièrement à une application ou à un type de problème.

Méta-Tâche : une méta-tâche représente un ensemble de tâches permettant d'atteindre le même but.

Plan d'Actions : un plan d'actions est un ensemble de tâches élémentaires dont l'exécution permet de résoudre effectivement le problème posé.

Requête : une requête, composée d'un but à atteindre (choisi parmi un ensemble prédéfini de buts) et d'un ensemble de données à traiter, est destinée à permettre à l'utilisateur (ou au système) de demander la résolution d'un problème en particulier.

Séparation de Sources : dans de nombreux domaine, on utilise les observations extérieures d'un phénomène – i.e. les signaux reçus sur un réseau de capteurs – pour caractériser leur cause – un ou plusieurs émetteurs qui sont à l'origine des signaux observés –. La séparation de sources vise à retrouver, le plus précisément possible, à partir des signaux observés – qui sont, en fait, la superposition des signaux émis par les différentes sources de la manifestation observée -, les sources qui en sont à l'origine.

Société de Spécialistes : la « société de spécialiste » est un modèle multi-agents hybride qui combine les avantages des systèmes à tableau noir et des systèmes d'acteurs : à un premier niveau d'abstraction des regroupements d'agents - communicant par le biais d'un tableau noir – sont effectués par pôle de compétence ; ces différents groupes de spécialistes collaborent ensuite les uns avec les autres par envoi de messages à un second niveau d'abstraction afin de résoudre le problème en cours de traitement.

Sous-Tâche : une sous-tâche est une tâche intervenant dans la stratégie d'une autre tâche ; on peut rencontrer deux types de sous-tâches : des sous-tâches qui sont une spécialisation d'une autre tâche (alors qualifiée de tâche générique) et des sous-tâches qui entrent dans la décomposition fonctionnelle d'une autre tâche.

Spécialisation de Tâche : la spécialisation de tâche est l'action qui consiste, à partir d'une tâche générique, à choisir la tâche la plus adaptée au problème abordé, i.e. la plus spécifique. Dans un formalisme objet, la spécialisation est directement calquée sur le mécanisme d'héritage de classes (c.f. ***SCARP*** & ***AROME***). Ajoutons que dans le cadre du modèle développé durant ce travail, la spécialisation consiste, en fait, en un choix dépendant uniquement du type des entrées/sorties de la requête à résoudre et de la tâche qui doit y parvenir.

Stratégie : une stratégie définit la manière de résoudre un problème en décrivant les étapes qui composent sa résolution ; autrement dit, une stratégie permet de spécifier « comment » décomposer le problème abordé en sous-problèmes plus simples à résoudre.

Système à Bases de Connaissances (SBC) : un système à bases de connaissances est un système capable de réaliser des fonctions de raisonnement symbolique en s'appuyant sur une représentation adéquate des connaissances mise en jeu et sur des mécanismes efficaces d'exploitation de ces connaissances ou de raisonnement.

Système à Tableau Noir : un système à tableau noir est un système multi-agents dans lequel les agents communiquent par le biais d'une zone de mémoire partagée appelée « tableau noir ».

Système Multi-Agents (SMA) : un système multi-agents est un système basé sur le partage et la distribution entre plusieurs agents (intelligents ou non) de l'ensemble des connaissances et la capacité de raisonnement que possède un système intelligent.

Tâche : une tâche représente une classe de problèmes (réalisation d'un but) à travers des données d'entrée et de sortie, ainsi qu'une stratégie pour le résoudre.

Tâche Abstraite (ou Générique ou Spécialisable) : une tâche abstraite est une tâche dont le niveau de description est volontairement général de sorte à regrouper un ensemble de tâches qui représentent des problèmes ayant des caractéristiques communes. La stratégie de résolution d'une telle tâche est un choix parmi un ensemble de tâches « plus concrètes » - qui en sont des cas particuliers - en fonction du contexte du problème en cours de résolution.

Tâche Complexe (ou Composée) : une tâche complexe est une tâche qui représente un problème dont la stratégie de résolution est composée de plusieurs étapes combinées à l'aide d'opérateurs de composition ; le problème associé à une telle tâche est résolu par un plan d'action, c'est-à-dire, en fin de compte, une séquence de tâches élémentaires dynamiquement adaptée par le moteur de pilotage (ou plus précisément son module de planification). Résoudre une telle tâche revient donc à construire dynamiquement et à parcourir son arbre de décomposition/spécialisation.

Tâche Primitive (ou Elémentaire) : une tâche primitive est une tâche qui représente un problème dont la stratégie de résolution n'est composée que d'une seule étape, autrement dit, c'est une tâche « simple » dont la résolution est l'exécution de l'algorithme ou du module associé. Les tâches primitives correspondent donc aux feuilles des arbres de décomposition/spécialisation des tâches complexes.

S.C.D. - U.H.P.
NANCY 1
SCIENCES
BIBLIOTHÈQUE

INTRODUCTION

Présentation du problème et motivations

I. Généralités

A. L'origine du projet

L'interprétation de signaux est une activité courante dans de nombreuses applications pratiques telles que la surveillance ou la conduite de procédés, le contrôle de qualité ou encore la médecine. Cependant, l'interprétation proprement dite des signaux est le plus souvent confiée à un ou plusieurs opérateurs humains qui doivent être fortement spécialisés (ce sont des experts du domaine, en général) et ce tout en étant capables de maintenir leur attention pendant de très longs moments et sur des quantités de données plus qu'importantes.

C'est pour ces multiples raisons qu'il est capital d'assister l'humain dans cette opération en automatisant tout ou partie du processus d'interprétation. Bien que souvent basés sur l'utilisation des mêmes algorithmes classiques de traitement du signal, la conception de tels systèmes d'interprétation automatique de signaux pose de nombreux problèmes tels que le choix des algorithmes à utiliser, leur ordre d'exécution ou la prise en compte de la nature des signaux à traiter en cours d'interprétation.

Dans le cadre de cette phase de développement de systèmes d'interprétation de signaux, un des objectifs majeurs pour EDF est la réduction des développements informatiques de nouveaux modules algorithmiques, la réutilisation «intelligente » de composants logiciels existants et par-là même l'augmentation de la sûreté des logiciels développés à partir de ces derniers. C'est dans ce cadre général que se situe le travail effectué dans le cadre de cette thèse, en collaboration avec le Groupe Traitements Avancés de l'Information (TAI) du Département Surveillance, Diagnostic, Maintenance (SDM) en matière de développement d'outils de capitalisation des développements et des savoir-faire.

La conception d'un système générique d'aide à l'utilisation et à la mise en œuvre d'algorithmes numériques - qui peut être vu comme une bibliothèque « intelligente » - est un projet ambitieux qui s'inscrit directement dans la droite ligne de cette action. Il s'agit, à partir d'une bibliothèque de modules numériques, de disposer d'une « surcouche » permettant à un utilisateur d'être guidé dans le choix d'une méthode parmi plusieurs en compétition, d'identifier la plus adaptée à ses données, d'être aidé dans la mise en œuvre de la méthode et éventuellement dans l'exploitation et/ou l'interprétation des résultats obtenus : afin de contribuer à résoudre le problème de la capitalisation des méthodes et des savoir-faire, le travail à réaliser consiste à définir et à tester un ***Environnement d'Aide au Développement***

basé sur une bibliothèque de programmes et sur l'utilisation de techniques de ***Pilotage de Programmes*** permettant de choisir et de piloter non seulement les algorithmes de traitement du signal, mais également les «agents» d'interprétation à utiliser afin de réaliser la tâche d'interprétation de signaux que l'on souhaite automatiser. Il s'agit donc de concevoir un modèle générique capable d'aborder une grande variété de problèmes (séparation de sources, analyses spectrales de signaux, etc.) et qui permette en outre de prendre en compte les incertitudes introduites par les méthodes de traitement du signal, ainsi que les aspects dynamiques liés à l'évolution des signaux dans le temps.

B. Environnement d'Aide au Développement, Pilotage de Programmes et Modèle de Tâches

B.1. Pourquoi définir un Modèle de Tâches ?

Un système coopératif d'aide au développement de systèmes d'interprétation de signaux – qui de notre point de vue peut être considéré comme une « Bibliothèque de Programmes Intelligente » - doit être capable - de façon autonome et presque automatique - de choisir les modules algorithmiques à utiliser et d'organiser leur exécution de façon à réaliser la tâche désirée, mais il doit aussi permettre à son utilisateur d'intervenir - de façon plus ou moins ponctuelle - dans ces processus de choix et d'organisation des exécutions. Pour permettre au système et à son utilisateur de coopérer, le raisonnement du système doit être facilement compréhensible par l'utilisateur, et inversement, l'utilisateur doit pouvoir aisément communiquer son propre raisonnement et ses propres connaissances au système. Pour cela, il convient de proposer une modélisation des connaissances et du processus de résolution de problèmes – car, dans ce cadre, concevoir une nouvelle application à partir d'éléments logiciels de base préexistants revient à résoudre le problème suivant : comment réaliser telle tâche complexe avec les tâches de base dont on dispose ? –. Le formalisme de modélisation des connaissances qui nous a semblé le plus adapté à notre problèmatique est celui des ***« Modèles de Tâches »***. Un modèle de tâches – construit autour d'une entité : la tâche – est un modèle basé sur le paradigme suivant : « résoudre un problème, c'est effectuer un certain nombre de tâches élémentaires qui permettent d'atteindre le but associé au problème traité ». Un modèle de tâche précise donc la structure des tâches – une tâche étant alors associée à une classe de problèmes – et les moyens de les composer afin de pouvoir aborder des problèmes de compléxités variables de façon modulaire (i.e. résoudre un problème complexe revient le plus souvent à résoudre un ensemble de sous-problèmes plus simples).

B.2. Pourquoi utiliser le Pilotage d'Algorithmes ?

La conception d'un système d'interprétation automatique de signaux - à partir d'une bibliothèque de programmes prédéfinis - et l'adaptation dynamique des traitements à effectuer au fur et à mesure de l'interprétation en cours d'élaboration peuvent être traités d'une manière

identique si l'on adopte une approche unifiée basée sur le ***Pilotage d'Algorithmes***. En effet, la résolution de l'un ou l'autre de ces deux problèmes revient le plus souvent à choisir des modules algorithmiques - parmi un ensemble -, puis à enchaîner leur exécution.

Pour que cela soit possible, il convient de fournir au système de pilotage une représentation uniforme et adaptée des différents algorithmes manipulés ; c'est donc à ce niveau qu'intervient notre choix de retenir une représentation basée sur un modèle de tâches. Ainsi, nous avons défini un modèle de tâches adapté à nos besoins, celui-ci devant, en effet, permettre de décrire, d'une part les modules algorithmiques en termes des objets qu'ils acceptent en entrée et produisent en sortie, d'autre part les enchaînements de ces modules (nous parlerons alors de «stratégies ») en vue de la résolution d'une classe de problèmes. La résolution proprement dite d'un problème consiste alors à exécuter la tâche associée en la décomposant récursivement et de façon opportuniste (c'est-à-dire en tenant compte des résultats intermédiaires obtenus) en sous-tâches plus élémentaires jusqu'à l'exécution directe de procédures.

B.3. Notre Modèle de Tâches en quelques mots

Le modèle de tâches que nous avons tenté de définir dans ce document s'inspire du paradigme suivant : l'exécution d'une tâche complexe peut être vue comme l'exploration dynamique d'un arbre de décomposition/spécialisation de cette même tâche. Ainsi, un système d'interprétation de signaux automatique - et plus généralement un système informatique réalisant une tâche de haut niveau - peut être considéré comme un système exécutant un programme complexe composé de plusieurs modules algorithmiques - plus simples – dont l'enchaînement est prédéterminé – avant l'exécution – en fonction de la tâche complexe à effectuer (c'est-à-dire en fonction du contexte d'application des différentes tâches la composant) et est ensuite dynamiquement adapté – au cours de l'exécution – suivant le contexte de l'exécution. Cette distinction permet de faire apparaître naturellement deux niveaux de contrôle – au sein du mécanisme de pilotage de programmes - qu'il convient de distinguer :

1. Le niveau statique qui, en fonction de la classe de problème abordée, permet d'établir une solution type – un arbre de tâches permettant de résoudre la tâche complexe visée – composée d'un enchaînement plus ou moins complexe de modules algorithmiques simples, directement exécutables ;

2. Le niveau dynamique qui, suivant le contexte d'exécution (i.e. la qualité des résultats déjà produits, la nature ou la qualité des données à traiter, etc.) permet

d'adapter au fur et à mesure les traitements à effectuer afin de réaliser la tâche complexe initialement visée.

1. Les enjeux

Les enjeux qui se cachent dérrière notre travail - qui sont d'ailleurs essentiellement d'ordre économique - sont ceux que l'on retrouve en génie logiciel derrière le terme générique de ***« réutilisabilité »***. En effet, basée elle aussi sur la modularité des composants logiciels (développement de packages et/ou de librairies logiciels) et sur les principes d'abstraction et de généralisation, la réutilisabilité - qui est le fait de pouvoir réutiliser tout ou partie d'un logiciel existant en incluant certains de ces composants dans de nouveaux développements - a pour objectifs de faciliter la tâche des concepteurs et des développeurs d'applications informatiques en leur permettant de disposer des connaissances et savoir-faire déjà acquis dans le domaine.

A. La réutilisation : un gain de temps et d'argent

Basée sur le caractère répétitif de la programmation, la réutilisation de composants logiciels existants a pour objectif premier de permettre aux développeurs de réutiliser l'existant à chaque fois que cela est possible afin de leur permettre de gagner en productivité (réduction des coûts et des temps de développement) et en réactivité.

B. La réutilisation : un gain en terme de sûreté

En outre, la réutilisation permet, du fait qu'elle rend possible le test et la validation des composants utilisés par un plus grands nombre de personnes, d'accroître la sûreté des logiciels développés. En effet, un logiciel basé sur la réutilisation de composants déjà validés nécessitera la mise en œuvre de beaucoup moins de moyens pour être à son tour validé.

C. La capitalisation des savoir-faire

Un autre aspect essentiel de notre démarche est de vouloir permettre la capitalisation et le partage des connaissances expertes. En effet, cette préoccupation partagée par de nombreux laboratoires de recherches et par de nombreuses entreprises, repose sur la constatation suivante : former un expert est une tâche longue et difficile et la transmission (ou transfert) des connaissances est souvent une chose ardue. Ainsi, pour éviter les grandes « fuites » des connaissances et savoir-faire liées aux flux migratoire (mutations, retraites, etc.), il est essentiel de mettre en place des structures permettant l'acquisition et le stockage des connaissances expertes, ainsi que de celles nécessaires à leur bonne utilisation.

D. L'aide au développement

Enfin, la réutilisation de composants logiciels combinée à une volonté de capitalisation des connaissances expertes et des savoir-faire permet d'entrevoir un moyen sûr d'offrir à des utilisateurs non experts d'un domaine un outil de conception, de maquettage et de test de nouvelles applications. Ainsi, on peut voir dans notre travail la volonté de proposer, par le biais d'une ***« bibliothèque de programmes intelligente »***, un environnement d'aide au développement.

2. Plan du mémoire

Ce mémoire est organisé de la façon suivante :

- le **chapitre 1** présente tout d'abord au lecteur la problématique des environnement de résolution de problèmes en tentant de dresser un résumé succinct des principaux thèmes de recherches qui ont émergé dans ce domaine ces dernières années ; ensuite, il insiste sur l'aspect pluridisciplinaire du problème abordé et sur les apports des différentes communautés scientifiques concernées ; Puis un rapide rappel sur les modèles de tâches et sur le paradigme qui est à leur origine est dressé. Enfin, le lecteur y trouvera une présentation des différentes approches et architectures de ces dernières années qui ont retenu notre attention : les différents modèles de tâches présentés à cette occasion ont été regroupés selon qu'ils ont pour vocation d'être conceptuels, opérationnels ou hybrides.

- le **chapitre 2** est consacré à la présentation de GALAXIE, notre Environnement d'Aide à la Conception et au Maquettage d'Applications de Traitement du Signal et d'Interprétation de Signaux. Dans une première partie, nous présenterons le modèle de tâches que nous avons été amené à définir et qui se veut être une synthèse étendue des différents modèles présentés au chapitre 1. Dans une seconde partie, nous présenterons notre moteur de pilotage et son fonctionnement dans ses deux architectures possibles : centralisée et distribuée.

- le **chapitre 3** présente la maquette que nous avons réalisé afin de tester et de valider notre modèle de tâches et, de façon plus générale, notre approche par « pilotage de programmes » ; celle-ci s'attaque au problème de la séparation de sources et met en œuvre une version « allégée » de notre moteur de pilotage. Dans ce chapitre, nous présenteront également nos conclusions sur les résultats atteints par cette première maquette.

- le **chapitre 4**, enfin, clôturera ce mémoire en présentant les perspectives qui nous semblent les plus prometteuses et en présentant nos conclusions quant au travail réalisé dans le cadre de cette thèse.

S.C.D. - U.H.P.
NANCY 1
SCIENCES
BIBLIOTHÈQUE

Chapitre 1

ETAT DE L'ART

Environnement d'Aide au Développement,
Environnement de Résolution de Problèmes
et Bibliothèque de Programmes « Intelligente »

I. Introduction générale

A. Le contexte

L'informatique est devenue, en moins d'un quart de siècle, un outil incontournable pour les scientifiques et les ingénieurs. En effet, les avancées de l'informatique, tant sur le plan matériel que logiciel, de ces dernières décennies ont été si rapides que chacun peut désormais disposer aisément d'une puissance de calcul considérable ; cela a donc permis aux ingénieurs et aux scientifiques d'avoir systématiquement recours au prototypage informatique pour concevoir et tester, à moindres frais, de nouveaux produits ou encore pour étudier, modéliser et simuler des phénomènes jusqu'alors trop complexes pour être abordés par l'homme. Malheureusement, ceux qui souhaitent utiliser ces techniques sont encore, à ce jour, dans l'obligation d'avoir des compétences, non seulement dans leur propre domaine d'activité, mais aussi dans des domaines aussi variés que les mathématiques appliquées, l'analyse numérique, l'algorithmique, ou encore plus généralement l'informatique (connaissance des multiples plates-formes matérielles et logicielles, des différents systèmes d'exploitations, des divers langages de programmation, etc.). La situation risque encore d'empirer avec l'avènement de modèles de plus en plus complexes qui font appel à des techniques variées issues de domaines très différents (mathématiques, statistiques, physiques, etc.) ; la conception et l'optimisation de tels systèmes multidisciplinaires nécessitent donc un niveau d'intégration des modèles et des systèmes bien supérieur à celui actuellement utilisé. Afin de répondre à ce besoin grandissant, il est devenu nécessaire de concevoir de nouveaux environnements, plus puissants, pour la modélisation et la simulation ; ces derniers devront aider leurs utilisateurs dans les phases de conception, de prototypage, de maquettage et de mise en œuvre des solutions logicielles dont ils ont besoin. Ainsi, il semble essentiel de fournir à ces derniers des outils génériques de résolution de problèmes qui puissent les aider à résoudre rapidement, facilement et efficacement un problème donné - tout en minimisant les temps et les coûts de développement et en assurant une certaine qualité (sûreté) des solutions développées -, soit en combinant des modules logiciels préexistants, soit en leur permettant de développer de nouveaux modules à partir de ceux dont ils disposent déjà.

C'est dans ce cadre que les ingénieurs et les scientifiques ont, dans un premier temps, commencé à constituer des bibliothèques de programmes, ensemble de modules algorithmiques destinés à permettre la réutilisation de composants logiciels (avec tous les enjeux associés : gain de productivité, de sûreté, etc.), le transfert technologique (i.e. utilisation industrielle de modules algorithmiques directement issus des fruits de la recherche) et l'adaptation (possibilité de choisir la méthode la plus adaptée au contexte parmi un ensemble de méthodes concurrentes). Cependant, un tel procédé ne permet que la capitalisation de méthodes et ignore totalement celle des savoir-faire qui sous-tendent leur bonne utilisation : le nombre croissant et la complexité grandissante des algorithmes disponibles nécessitent de mettre en place des procédés complexes, mais uniformisés, de description et d'indexation ; de même, l'utilisation d'algorithmes de plus en plus spécifiques suppose d'avoir l'expertise nécessaire dans les phases de choix et d'utilisation d'un algorithme parmi un ensemble d'algorithmes concurrents.

Ainsi, dans un second temps, la communauté scientifique (les mathématiciens, les experts en traitement du signal ou encore les informaticiens) s'est intéressée aux Environnements de Résolution de Problèmes (ERP), environnements dédiés à un ou plusieurs domaines, dans lesquels les utilisateurs sont guidés dans la résolution des problèmes qu'ils souhaitent aborder ; ces environnements de résolution de problèmes sont donc des systèmes informatiques complexes qui doivent aider leurs utilisateurs à planifier et à contrôler l'exécution d'actions leur permettant de résoudre une catégorie pré-déterminée de problèmes en mettant à leur disposition tous les outils nécessaires à cette résolution (notamment une bibliothèque d'algorithmes solutions aux problèmes classiques, des méthodes de choix d'algorithmes en fonctions des caractéristiques du problème posé, des moyens d'étoffer la bibliothèque existante par ajout de nouveaux algorithmes, des moyens d'évaluation des solutions construites).

L'intérêt essentiel de tels environnements, outre la possibilité de mettre en œuvre le concept, cher aux informaticiens, de réutilisation de composants logiciels pré-existants, est de permettre à leurs utilisateurs de résoudre un certain nombre de problèmes complexes sans avoir besoin de connaissances spécialisées en informatique ou sur les méthodes utilisées (condition d'utilisation, etc.).

Les principes sous-jacents à de tels environnements est la capitalisation des savoir-faire et l'accès facile à des méthodes très pointues ; comme l'écrit J.R. Rice dans **[Rice 96]**, « ils doivent offrir tout à tout le monde » : ils doivent pouvoir résoudre des problèmes de complexités variables, permettre le prototypage rapide tout comme l'analyse détaillée, être utilisables par des novices tout autant que par des spécialistes du domaine.

Les premiers environnements de résolution de problèmes étaient constitués de petits ensembles de modules tirés de bibliothèques de programmes existantes et destinés à la résolution de problèmes mathématiques ou d'ingénierie prédéfinis (parmi les plus célèbres, citons Mathematica et Matlab pour les mathématiques et le traitement du signal) ; ceux vers lesquels il nous semble falloir converger devront gagner en généricité et permettre ainsi d'aborder un nombre varié de problèmes différents. Pour résumer les choses, on peut définir un environnement de résolution de problèmes par ses composants :

ERP = Interface Utilisateur + Bibliothèques de Programme
+ Bases de Connaissances + Intégration

Les concepts sur lesquels reposent ces environnements génériques de résolution de problèmes - qui se trouvent à l'intersection de deux disciplines fondamentales de l'informatique moderne, le « Génie Logiciel » et « l'Intelligence Artificielle » - se retrouvent aussi dans les fondements du pilotage d'algorithmes (choix contextuels de méthodes, surveillance d'exécutions, évaluation de performances, « réparation » des solutions proposées, tout cela organisé autour d'une bibliothèque de programmes), ce qui justifie en partie le rapprochement que nous avons opéré entre techniques de pilotage de programme et environnement de résolution de problèmes.

La capitalisation des logiciels et des savoir-faire associés à leur « bonne » utilisation est donc un enjeu capital, non seulement dans le domaine de la recherche – pour laquelle aucune avancée n'est envisageable sans capitalisation -, mais aussi dans le domaine de l'ingénierie, qu'il s'agisse d'informatique, de mathématiques ou de tout autre discipline pouvant bénéficier des services offerts par l'outil informatique. Ainsi, comme tente de l'illustrer la figure suivante, cette problématique est au centre des travaux de différentes communautés scientifiques : pour la communauté de « Traitement du Signal » c'est un enjeu capital en terme de gain de productivité et de capitalisation de méthodes et des savoir-faire, tandis que pour la communauté « Génie Logiciel », l'accent est porté sur l'aspect aide au développement et sûreté des développement (dans les travaux actuel [Lévy 94], [Lambolais 98], l'important étant le plus souvent d'aider à concevoir automatiquement des logiciels sûrs dont la conception s'accompagne d'une justification automatique en terme de sûreté via les « preuves de programmes » associées) ; pour la communauté d'« Intelligence Artificielle », à laquelle nous appartenons, il s'agit plus directement de mettre en place des mécanismes de résolution de problèmes plus ou moins génériques.

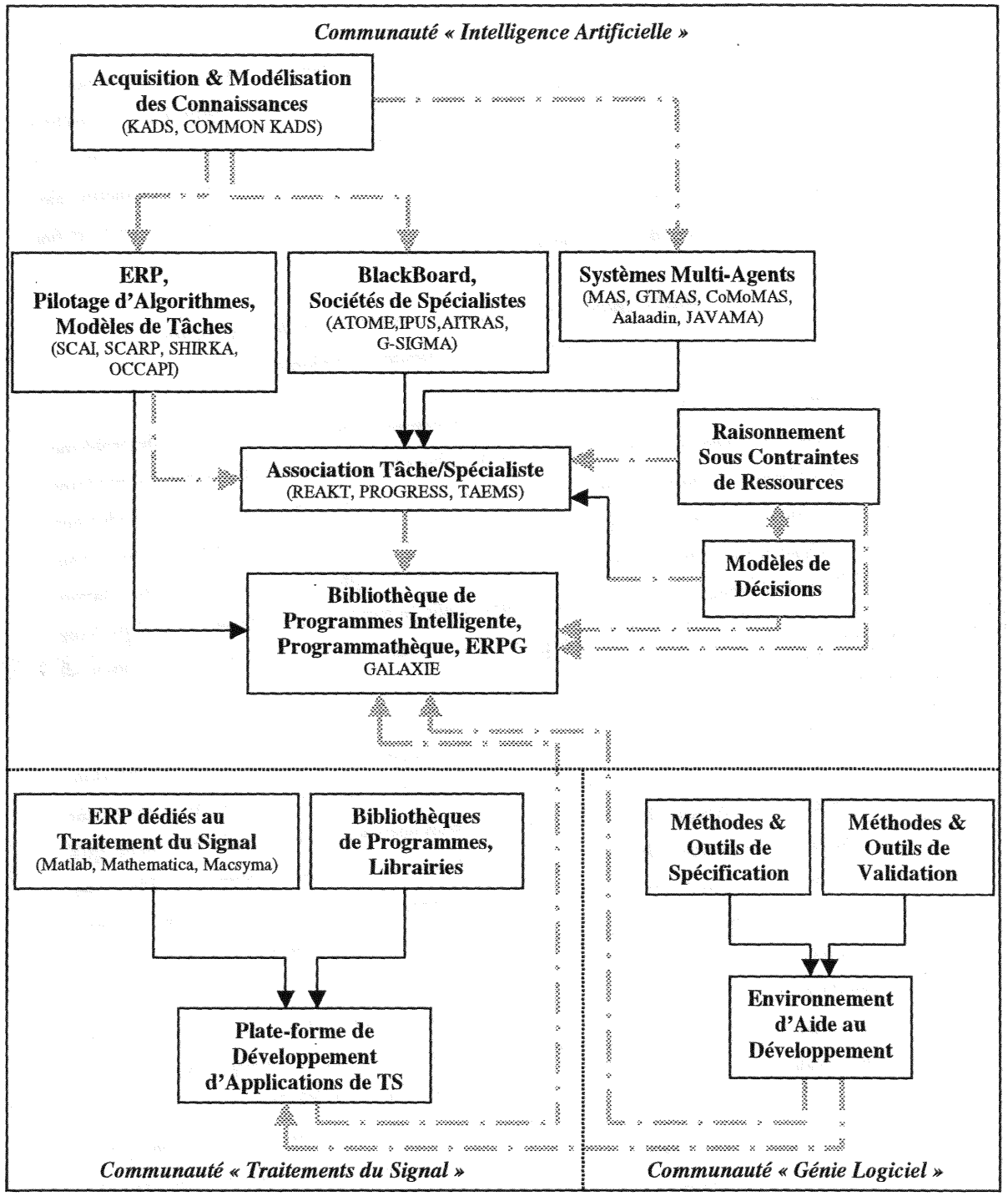


Fig. 1 : Une préoccuption issue de plusieurs communautés

B. Les modèles de tâches

B.1. Remarques préliminaires

Les modèles de tâches ont été utilisés dès leur introduction par B. Chandrasekaran, au début des années 80, pour faciliter l'élaboration de systèmes à bases de connaissances (SBC) destinés

à la résolution de problèmes complexes : l'analyse d'une tâche doit conduire à la description de sa décomposition en sous-tâches de moindre complexité et à la définition des connaissances qui sont nécessaires à son exécution. La décomposition successive en sous-tâches moins complexes doit permettre d'aboutir à des tâches directement réalisables (qu'on appellera alors tâches élémentaires), c'est-à-dire associées à une action (intervention humaine, exécution d'un module logiciel, inférence du SBC).

Parallèlement à cela, les modèles de tâches ont été utilisés pour modéliser les connaissances nécessaires au développement d'environnement de résolution de problèmes dédiés aux mathématiques ou à la physique et organisés autour de bibliothèques de programmes.

Dans le cadre de ces environnements de résolution de problèmes, les modèles de tâches se sont complexifiés et ont ainsi progressivement été utilisés pour représenter des raisonnements permettant de résoudre des classes de problèmes complexes en tenant compte des caractéristiques de chaque problème, ce qui inclus non seulement des informations sur les problèmes eux-mêmes (modélisation du domaine, description des solutions types), mais aussi informations sur les différentes méthodes utilisables au cours de la résolution. Ainsi, on s'est intéressé non plus seulement à la capitalisation des méthodes, mais aussi à celle des savoir-faire.

B.2. Qu'est-ce-qu'une Tâche

Le mot ***« tâche »*** a été utilisé de diverses façons dans notre domaine de recherche, contribuant ainsi à rendre certains exposés confus. Par exemple, Wielenga (dans [Wielinga 92]) définit une tâche comme une « stratégie fixe » (ou statique) pour atteindre un but, ce qui signifie que c'est une sorte de spécification de méthode ou de procédure. Dans les premiers articles de Chandrasekaran sur les tâches génériques (cf. [Chandrasekaran 86]), il y avait confusion entre le but et la méthode : les tâches génériques pouvaient être vues comme des composants d'une méthode composée (ou complexe) (comme c'est le cas pour le but « diagnostiquer » qui est atteint par le biais d'une méthode composée d'une phase d'abstraction des données et d'une phase de classification), ou bien comme des buts eux-mêmes (la tâche générique « construction d'hypothèse » a pour but de construire des hypothèses à partir des entrées observées). Dans cet article de 1992 ([Chandrasekaran 92]), Chandrasekaran utilise le mot « tâche » comme synonyme de « types de buts de résolution de problème », c'est-à-dire, plus généralement, pour désigner une classe de problèmes : ainsi, on parlera d'une tâche de diagnostique afin d'abstraire la classe de problèmes qui consiste à atteindre le but suivant : générer une explication causale du comportement anormal observé.

B.3. Les différents types de modélisations basées sur les Tâches

Cette section décrit les deux grandes approches qui ont été adoptées dans les différents modèles de tâche que nous allons présenter par la suite. En effet, décrire une tâche revient à décrire sa décomposition au travers de sa stratégie associée. Pour ce faire, deux approches on été développées : la première repose sur le schéma de décomposition d'une tâche en méthodes associées, elles-mêmes décomposables en sous-tâches ; la seconde repose plus directement sur la décomposition d'une tâche en sous-tâches.

Décomposition Tâche/Méthode/Sous-Tâche

Dans cette approche, l'énoncé d'un problème et la description du moyen de le résoudre sont découplés [Chaillot 93, Chandrasekaran 84, Chandrasekaran 86, Chandrasekaran 89, Chandrasekaran 92, Chandrasekaran 93, Chandrasekaran 97, Musen 89, Musen 93]. Dans cette représentation, le mot « tâche » est synonyme de « type de but de résolution de problèmes » , c'est-à-dire qu'il désigne une classe de problèmes : ainsi, on parlera d'une tâche de diagnostic afin d'abstraire la classe de problème qui consiste à atteindre le but suivant : générer une explication causale du comportement anormal observé. A chaque tâche est alors associée une ou plusieurs méthodes qui permettent sa résolution (i.e. l'obtention du but poursuivi) ; chacune de ces méthodes est caractérisée par les formes de connaissances et d'inférences qu 'elle nécessite et par les sous-buts additionnels (ou sous-problèmes) qui ne seront pas directement atteints (résolus) par son exécution et qu'il conviendra d'atteindre (de résoudre) afin de compléter la résolution du problème en cours de traitement. Cette spécification de nouveaux sous-buts se fait en associant aux méthodes des décompositions en sous-tâches qui récursivement renverront à leur tour à de nouvelles méthodes, et ainsi de suite.

Dans cette approche, comme décrit dans [Chandrasekaran 89], une structure de tâche est une représentation de la tâche en terme de méthodes applicables pour la réaliser dans un domaine particulier et en terme de conditions d'application des différentes méthodes. Chaque méthode est décrite par le biais des connaissances et des mécanismes qu'elle utilise et par le biais des sous-buts qu'elle ajoute et nécessite d'atteindre avant de se solder par un succès. Ce genre de décomposition peut être fait récursivement jusqu'à obtention de méthodes qui permettent d'atteindre tous les sous-buts en attente, sans en ajouter de nouveaux (ce sont des méthodes directement exécutables qui sont parfois dénommées ***mécanismes*** (***mecanisms***)

S.C.D. - U.H.P. NANCY 1 SCIENCES BIBLIOTHÈQUE

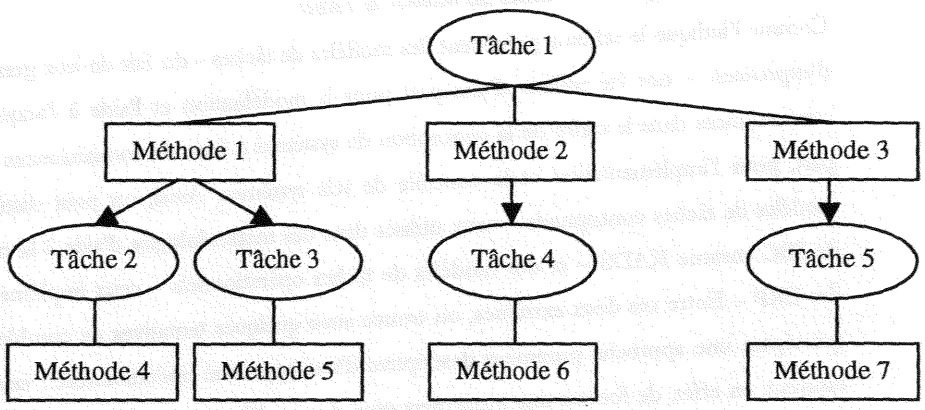


Fig. 2 : L'approche Tâche/Méthode/Sous-Tâche

Sur la figure ci-dessus, la tâche 1 est associée à trois méthodes qui permettent de la réaliser (ces méthodes sont donc en concurrence), elles-mêmes décomposées en sous-tâches et récursivement jusqu'à obtention de méthodes directement exécutables : ainsi, résoudre la tâche 1 revient à exécuter, soit la séquence composée des méthodes 4 et 5, soit la méthode 6, soit la méthode 7. On s'aperçoit ainsi que la spécification de plusieurs méthodes pour une tâche permet de réaliser un **OU** entre ces dernières, tandis que la spécification de plusieurs tâches pour une méthode correspond à réaliser un **ET** entre ces dernières.

Décomposition Tâche/Sous-Tâche

Dans cette approche, l'énoncé d'un problème et la description du moyen de le résoudre sont indissociables et sont réalisés au moyen d'une seule et même structure : la tâche [Moisan 97, Parmentier 98, Parmentier 99, Willamowski 92a, Willamowski 92b, Willamowski 94, Wielinga 92]. Cette structure inclut donc – contrairement à ce qui est fait dans l'approche Tâche/Méthode/Sous-Tâche où la description du problème est séparée de celle du moyen associé à sa résolution – la description de la voie à suivre afin de résoudre le problème posé : c'est la ***stratégie*** dans SCARP [Willamowski 94] et dans notre modèle, le ***corps*** dans TASK ou le ***corps de la tâche*** (***task body***) dans CommonKADS [Wielinga 92] et dans Power-Tasks [Parmentier 98]. Ainsi, la stratégie - comme nous la désignerons maintenant – permet de décrire la décomposition en sous-tâche de la tâche considérée ; chacune de ses sous-tâches pouvant à leur tour se décomposer en sous-tâches, jusqu'à obtention de tâches directement exécutables (équivalentes aux méthodes directement exécutables, ou mécanismes, de l'approche Tâche/Méthode/Sous-Tâche).

Au sein de ce courant, plusieurs approches existent cependant, comme nous le verrons dans la suite de cette partie.

B.4. Les différents types d'utilisations des Modèles de Tâches

Comme l'indique le schéma précédent, les modèles de tâches - du fait de leur grand pouvoir d'expression – ont été utilisés d'une part pour la modélisation et l'aide à l'acquisition des connaissances dans le cadre de la conception de systèmes à base de connaissances et, d'autre part, pour l'implémentation et le contrôle de tels systèmes. Ainsi, on peut distinguer des modèles de tâches conceptuels – ceux utilisés dans des méthodologies d'aide à la conception de SBC comme KADS – et des modèles de tâches opérationnels – ceux implémentés dans des ERP -. Entre ces deux extrêmes, on trouve aussi quelques tentatives de combler le fossé qui sépare une approche purement descriptive d'une approche opérationnelle : ces modèles essayent, en effet, de fournir une implémentation des modèles préalablement décrits : on dira qu'ils tentent d'opérationnaliser une modèle conceptuel.

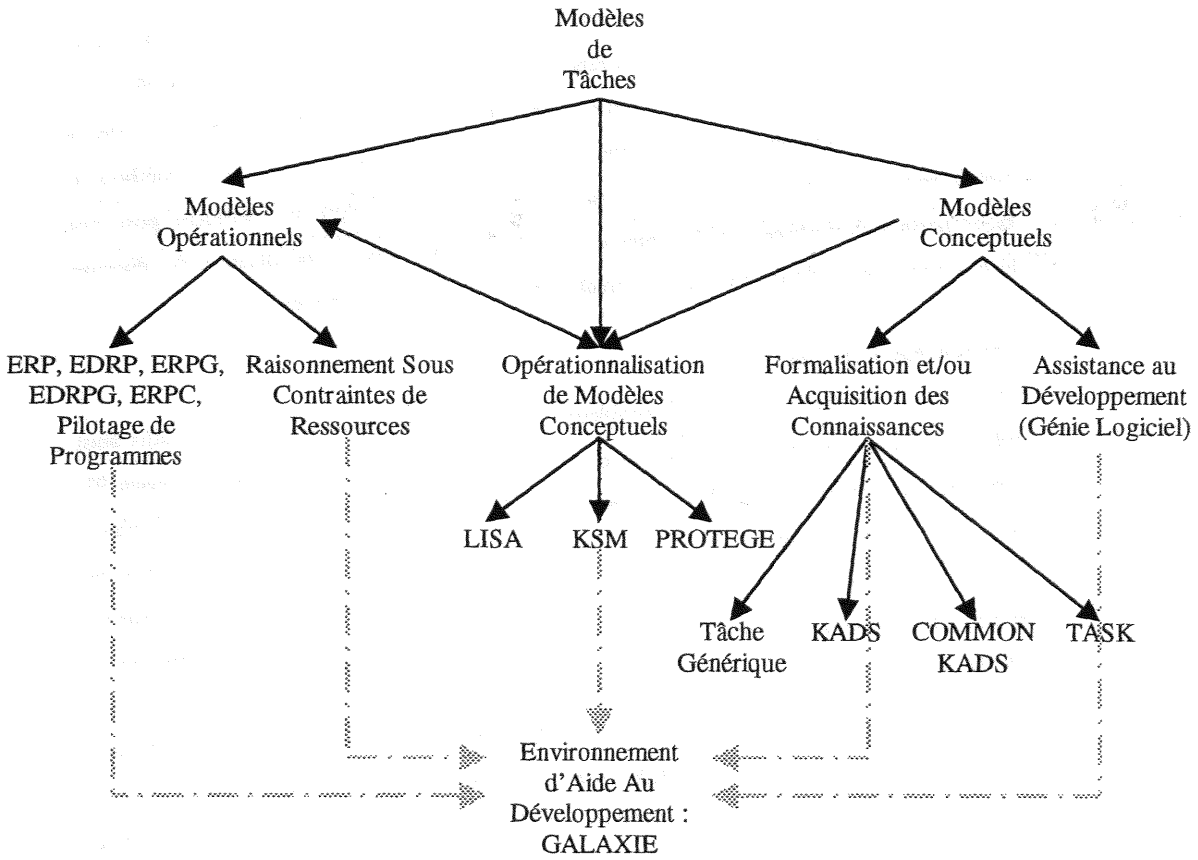


Fig. 3 : Modèles de Tâches Conceptuels Versus Opérationnels

II. Les modèles conceptuels

J'ai choisi de rassembler sous la dénomination « modèles conceptuels » les modèles dont la vocation est plutôt orientée vers la conception de systèmes à base de connaissances.

A. Les tâches génériques de B. Chandrasekaran

Dans [Chandrasekaran 89] Chandrasekaran définit l'essence des tâches génériques par les propos suivants :

- ***Problèmes, méthodes et sous-problèmes :*** Un problème (ou un but à résoudre) peut avoir une ou plusieurs méthodes associées à sa résolution (ou à l'obtention du but). Chacune des méthodes est caractérisée par les formes de connaissances et d'inférences qu'elle nécessite et par les sous-buts additionnels (ou sous-problèmes) qui ne seront pas atteints (résolus) par son exécution et qu'il conviendra d'atteindre (de résoudre) afin de compléter la résolution du problème en cours de traitement. (Une méthode peut-être une procédure si la séquence d'actions à effectuer est entièrement prédéfinie, ou cela peut-être plus abstrait : cela peut-être une recherche de solution).

- ***Des tâches génériques comme combinaison de problème / méthode / connaissances / inférence à la base de packages :*** les tâches génériques sont des tâches récurrentes dans le cadre de la résolution de problèmes. Ainsi, on peut capitaliser les savoir-faire en encapsulant des méthodes de résolution de problèmes par le biais de ces tâches génériques. Elles permettent de fournir des primitives de résolution de problèmes réutilisables (formalisme de description de connaissance, moteur d'inférence, mécanisme de raisonnement, etc.).

- ***Un langage de haut niveau pour chaque tâche générique :*** L'approche par tâches génériques permet d'identifier un certain nombre de combinaisons « problème / méthode / connaissance / inférence générique » utiles, et doit fournir des outils permettant de les encapsuler afin de les définir et de les rendre utilisables comme des blocs de base dans le cadre de la résolution de problèmes plus complexes.

- ***Tâches génériques & Acquisition de connaissances :*** l'approche par tâches génériques permet au cogniticien d'associer des méthodes appropriées à certains problèmes et de trouver les connaissances et les mécanismes de raisonnement qui sont nécessaires pour celles-ci et pour accéder à d'autres tâches génériques qui peuvent être utiles pour la résolution de sous-problèmes. Ainsi l'approche par tâches génériques est une aide directe dans le processus d'acquisition des connaissances puisqu'il permet de focaliser le processus d'acquisition sur les connaissances opérationnelles (connaissances et mécanismes d'inférence) nécessaires à la résolution du problème (i.e. l'exécution de la tâche).

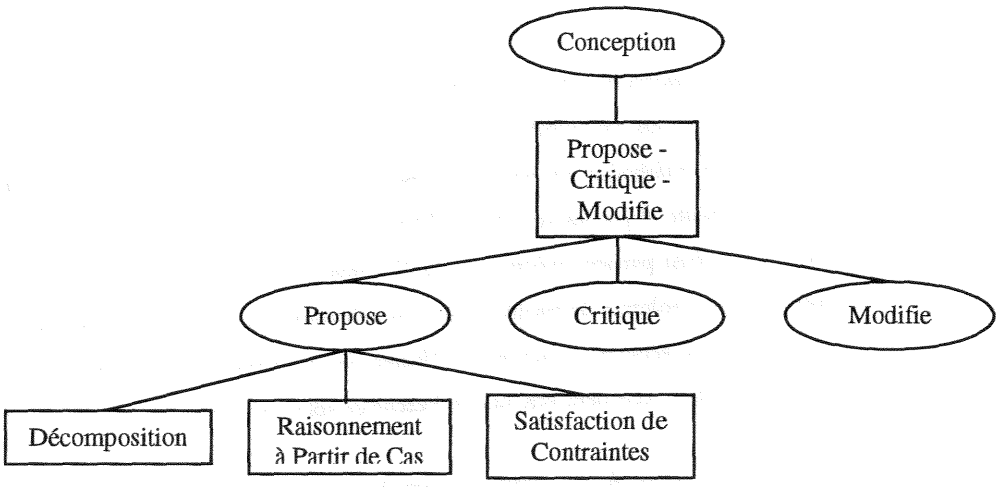


Fig. 4 : Description partielle de la tâche « concevoir » dans le formalisme de B. Chandrasekaran.

Remarque : Dans la figure précédente, les cercles représentent des tâches et les rectangles, des méthodes.

B. KADS et CommonKADS

KADS (pour ***Knowledge Acquisition and Design Support***) est une méthodologie d'aide à l'acquisition des connaissances dans le cadre de la conception d'un système à base de connaissance (ou **SBC**).

Le développement d'un **SBC** est défini dans ***KADS*** comme la composition d'un ensemble de modèles de différents types. ***KADS*** propose un ensemble de modèles types qui peuvent être complétés et raffinés durant la phase de développement, chacun de ces modèles représentant un aspect particulier de l'environnement ou du système et étant en relation avec les autres modèles. La méthodologie proposée dans ***KADS-I*** repose sur différents modèles : un modèle organisationnel, un modèle d'application et un modèle de tâches permettent la description du but poursuivi lors de la construction du **SBC** ; un modèle de coopération permet de décrire les fonctionnalités des tâches qui nécessitent de la coopération ; un modèle d'expertise permet de spécifier l'expertise de résolution de problèmes nécessaire à la réalisation des tâches de résolution de problèmes. Les modèles de coopération et d'expertise sont combinés dans un modèle conceptuel. Finalement, un modèle de design sert à décrire les techniques informatiques utilisées et permet de séparer la modélisation conceptuelle des aspects de d'implémentation.

KADS-I a été l'un des premiers projets de recherche européen dédié à la définition et au développement d'une méthodologie destinée à accompagner le concepteur d'un ***SBC*** dans les différentes phases de son cycle de développement. ***KADS*** définit le développement d'un **SBC** par la construction d'un ensemble de modèles à partir des modèles d'interprétation prédéfinis disponibles dans les librairies associées. Comme nous l'avons déjà mentionné, cet ensemble de modèles se structure en 6 modèles qui permettent de représenter les différents types de connaissances nécessaires lors de la conception d'un **SBC**. Le modèle de tâches, qui nous intéresse ici, n'est donc qu'un des aspects de cette méthodologie ; il consiste en une décomposition des tâches de la « vie courante » qui sont nécessaires pour atteindre le but du système tel qu'il a été spécifié dans le modèle d'application en un nombre fini de sous-tâches élémentaires et leur attribution aux différents agents (le système et ses différents utilisateurs). Il peut y avoir différentes façons d'atteindre le même but ; seules les caractéristiques de l'application, la disponibilité des connaissances et des données, et les contraintes externes imposées par l'utilisateur et par l'environnement permettent de déterminer l'alternative appropriée. C'est au cogniticien de déterminer les tâches internes et externes, c'est-à-dire celles qui seront à la charge du système et celles qui seront à la charge de l'utilisateur.

CommonKADS [Wielinga 92] est une méthodologie d'ingénierie des connaissances issue des projets **ESPRIT** (***KADS-I*** et ***KADS-II***) qui propose un nouvel ensemble de modèles pour la modélisation des connaissances. Là encore, nous ne présenterons que l'aspect « modèle de tâche » qui constitue que l'un des 6 modèles permettant la description exhaustive d'un **SBC**.

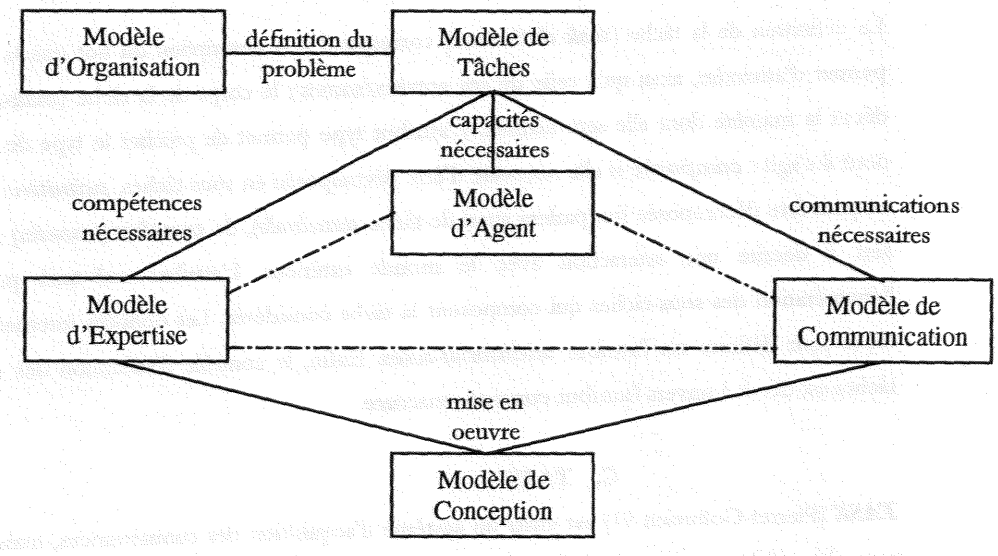


Fig. 5 : Composants de la méthodologie KADS (d'après [Wielinga 92])

Les connaissances manipulées dans ***CommonKADS*** sont de 2 types : les connaissances du domaines (dites connaissances statiques) et les connaissances de contrôle. Les premières sont

constituées de concepts, de relations et de faits concernant le domaine d'application concerné ; les secondes définissent la façon dont les connaissances du domaine doivent être exploitées afin de permettre la résolution des problèmes posés : c'est donc dans ces dernières que s'inscrivent les connaissances sur les tâches.

La spécification d'une tâche dans ***CommonKADS*** comporte 2 parties :

- la ***définition de la tâche*** (***task definition)*** qui décrit le but que permet d'atteindre la tâche, c'est-à-dire ce qu'elle doit réaliser ;

- le ***corps de la tâche*** (***task body***) qui décrit les activités nécessaires à l'accomplissement de la tâche, c'est-à-dire comment atteindre le but.

task <name>
    task-definition
        goal :description du but à atteindre
        input :
        output :
    task-body
        type : <composite> | <primitive> | <transfer>
        sub-tasks :
        additional-roles :
        control-structure :
end

Fig. 6 : Paramètres de définition d'une tâche dans ***CommonKADS***

La définition de la tâche (***task definition***) correspond à la description du but que la tâche permet d'atteindre, ainsi qu'à celle de ses entrées/sorties ; le corps de la tâche (***task-body***) décrit la manière dont elle sera réalisée. L'attribut **type** permet de préciser le type de tâche dont il s'agit : ***composite*** si elle nécessite d'être décomposée en sous-tâches, ***primitive*** si elle ne peut-être décomposée (on parlera aussi de tâche ***terminale***), de ***transfert*** (***transfer***) si elle sert à décrire une interaction avec le monde extérieur. L'attribut ***sub-task*** permet l'énumération des sous-tâches qui composent la tâche considérée. Les données internes à la tâche sont définies via l'attribut ***additional-roles***. Enfin, le contrôle d'exécution des sous-tâches est décrit à travers l'attribut ***control-structure***.

C. TASK

TASK [Pierret-Golbreich 91] est aussi un système d'acquisition des connaissances, mais il a pour objectif de permettre d'expliciter le processus de résolution d'un problème. Dans ***TASK***, une tâche est une entité structurée qui inclut toutes les informations relatives à la résolution d'un problème sous la forme d'une association ***contexte/traitement***. Les ***traitements***

S.C.D. - U.H.P. NANCY 1 SCIENCES BIBLIOTHÈQUE

servent à spécifier toutes les méthodes qui permettent d'accomplir une tâche donnée. La structure d'une tâche dans ***TASK*** est présentée sur la figure suivante.

partie contexte
état-initial
état-final
but
paramètres
partie exécutive
corps
partie relationnelle
liste-généraliations
liste-spécialisations
liste-coopérations

Fig. 7 : Structure d'une tâche dans ***TASK***

La ***partie contexte*** sert à caractériser le problème en en précisant l'état initial et final, le but et les paramètres. L'attribut ***corps*** de la ***partie exécutive*** sert à décrire la stratégie de résolution associée à la tâche spécifiée : il contient soit une action, soit un enchaînement de sous-tâches , soit un ensemble de tâches concurrentes parmi lesquelles il conviendra d'effectuer un choix. La ***partie relationnelle*** sert à décrire les relations qui relient la tâche en question aux autres tâches définies dans le SBC (ces relations pouvant être de 3 types différents : relation de spécialisation, de généralisation ou de coopération).

Dans ***TASK***, 3 types de tâches sont distingués en fonction du type de leur ***corps*** :

- les tâches ***élémentaires*** dont le ***corps*** est constitué d'une action ; il s'agit des tâches dont le niveau opérationnel est caractérisable par une entité procédurale insécable ;

- les tâches ***composées*** dont le ***corps*** est une structure prédéfinie ; ce sont les tâches dont le niveau opérationnel correspond soit à un enchaînement invariable et prédéterminé de sous-tâches, soit au choix d'une alternative parmi un ensemble de tâches concurrentes. Dans le premier cas, on parlera de tâches ***figées*** puisque la définition de l'enchaînement des sous-tâches est pré-établi et invariable (il s'agit d'une stratégie statique). Dans le second cas, on parlera de tâches ***alternatives*** et l'enchaînement sera, à contrario, dynamiquement défini lors de l'éxécution.

- les tâches ***abstraites*** dont le corps n'est pas prédéterminé ; elles correspondent à des tâches génériques et sont utilisées pour représenter au sein d'une même entité les caractéristiques communes relatives à une classe de problèmes. Elles servent donc à décrire un problème général indépendamment de toute stratégie de résolution

associée ; la stratégie adéquate est alors déterminée au cours de l'exécution en fonction du contexte.

Comme je l'ai déjà mentionné, les tâches élémentaires et les tâches figées correspondent à des stratégies statiques, tandis que les tâches alternatives et les tâches abstraites correspondent à des stratégies dynamiques qui sont établies, en fonction du contexte, au fur et à mesure que la résolution du problème posé progresse.

III. Les modèles opérationnels

Les « modèles opérationnels » sont les modèles qui ont pour but d'automatiser tout ou parti d'un processus de résolution de problème : ils servent le plus souvent à modéliser un processus plus ou moins complexe et son contrôle.

A. SCARP et ses applications

SCARP (***Système Coopératif d'Aide à la Résolution de Problèmes***) [Willamowski 92a, Willamowski 92b, Willamowski 94], développé au sein du projet **SHERPA** de l'**INRIA**, est un ***ERP*** générique dédié au pilotage d'algorithmes coopératif : il s'agit de piloter des bibliothèques de programmes en coopération avec l'utilisateur du système qui peut intervenir soit pour choisir la stratégie de résolution la plus adaptée au contexte, soit pour affecter des valeurs à des paramètres restés indéfinis. Différentes ***ERP*** coopératifs ont été développées à partir de ***SCARP*** : ***SLOT*** pour l'analyse de données ; ***Said*** pour le diagnostic vibratoire de plate-formes pétrolières en mer, développé à l'**IFREMER** de Brest ; ***MYOSIS*** pour l'aide au diagnostic médical dans le cadre de maladies neuro-musculaires, réalisé en collaboration avec le service d'électromyographie du C.H.U. de Grenoble.

Dans ***SCARP***, une tâche, respectivement une procédure, est décrite de la façon suivante :

```
{<tâche>     est-un      =       tâche ;
   &entrée   <entrée>    $un     objet ;
   &sortie   <sortie>    $un     objet ;
             stratégie   $défaut (séquence          <liste-de-tâches>)  |
                                 (choix             <liste-de-tâches>)  |
                                 (choix-utilisateur <liste-de-tâches>)
                                 ...
   &stache   <tâche-1>   $un     (contexte
                                     exec   $un     <tâche>  |
                                                    <procédure> ) }

{<procédure>  est-un      =       procédure ;
   &entrée    <entrée>    $un     objet ;
   &sortie    <sortie>    $un     objet ;
              nom-fct     $valeur <programme> }
```

Fig. 8 : Descriptions partielles d'une tâche et d'une procédure

L'attribut ***stratégie*** permet à l'utilisateur de définir les étapes de la résolution du problème associé à la tâche au moyen d'opérateurs spécifiques (***séquence***, ***parallèle***, ***choix***, ***choix-utilisateur***, ***itération***, ***paral-iter***, ***utilisateur***, ***spécialisation***) ; l'attribut ***exec*** permet de spécifier la façon dont ces différentes étapes de la résolution seront réalisées : chaque étape peut renvoyer soit directement à un exécutable (i.e. une ***procédure***) s'il s'agit d'une tâche ***élémentaire***, soit à une autre tâche qui est alors une sous-tâche de la précédente s'il s'agit d'une tâche ***complexe***.

Dans ***SCARP***, il existe plusieurs façon de décrire des tâches alternatives ou concurrentes - c'est-à-dire des tâches différentes qui permettent de résoudre le même problème - :

- par l'opérateur ***choix*** lorsque c'est au système qu'il incombe de choisir la tâche la plus adaptée au contexte de la résolution ;

- par l'opérateur ***choix-utilisateur*** lorsque ce choix est à la charge de l'utilisateur ;

- par l'opérateur ***spécialisation*** lorsque le choix de la méthode repose sur les hiérarchies de tâches construites par le cogniticien qui a développé l'application, c'est-à-dire quand le système doit retenir la tâche la plus spécifique (il s'agit en fait d'un choix effectué par le système en fonction des entrées/sorties de la tâche). En fait, lorsque le moteur de pilotage rencontre cet opérateur, il tente de classifier - à l'aide des mécanismes/automatismes d'héritage - l'instance du problème en cours de résolution de sorte à opter pour la stratégie la plus fine.

La résolution d'un problème passe donc par une alternance de phases de spécialisation et de décomposition.

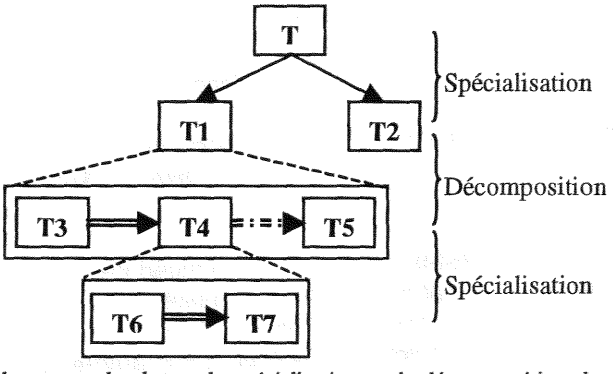


Fig. 9 : Alternance de phases de spécialisation et de décomposition dans ***SCARP***

Sur la figure précédente est représentée l'arbre de décomposition/spécialisation de la tâche ***T***: pour réaliser ***T***, on peut, suivant les caractéristiques des entrées/sorties de la tâche, opter pour ***T1*** ou ***T2***, deux spécialisations de ***T1***; pour réaliser ***T1***, il suffit d'utiliser sa décomposition telle qu'indiquée par sa stratégie associée : ici, la séquence de tâches ***T3***, ***T4***, ***T5***.

B. PowerTask

PowerTask [Parmentier 98, Parmentier 99] est un ***ERP*** générique coopératif développé par la société **ILOG** à partir des résultats issus des recherches menées par Jutta Willamowski dans le cadre de la définition de ***SCARP***. Cet outil générique de résolution de problème a été conçu autour d'un nouveau modèle de tâches orienté objet (ce qui permet, notamment, de tirer parti des mécanismes de classification automatique associés au concept d'héritage entre objets afin d'implémenter la spécialisation de tâches); il dispose d'un moteur de résolution associant à chaque tâche les méthodes possibles, et d'une interface graphique qui permet à l'utilisateur :

- de visualiser toutes les étapes d'une stratégie au cours de son exécution ;
- de spécifier les paramètres et les choix nécessaires à cette exécution ;
- de reprendre une exécution à n'importe quel endroit, ce qui lui permet d'émettre, de tester et de valider plusieurs hypothèses en parallèle ;
- de modifier une stratégie en modifiant ou en ajoutant des sous-tâches.

PowerTask a été utilisé dans le cadre du projet **GREG** pour développer un système coopératif d'aide à l'analyse de séquences génomiques (***ImaGene***) qui permet :

- de mettre en œuvre aisément des méthodes d'analyse de séquence :

- d'aider un utilisateur à choisir la ou les méthodes adéquates pour réaliser une tâche donnée (en les enchaînant si necéssaire dans le cadre de tâches complexes) ;

- de mémoriser, de gérer et de visualiser à la fois les données de l'analyse et les résultats produits par l'application de méthodes ;

- d'étendre le système au fur et à mesure en intégrant de nouvelles méthodes et leurs modes d'emploi.

Un travail de recherche, mené dans le cadre de la thèse de Thibault Parmentier, est actuellement en cours au sein du projet **SHERPA**, il vise à étendre l'architecture de ***PowerTask*** en y ajoutant des liens entre ressources et tâches de sorte à permettre une distribution des ressources et de leur utilisation.

C. TRAM

TRAM [Chaillot 93] est un ***ERP*** générique pour le pilotage de bibliothèques de programmes dans le cadre de systèmes réactifs devant prendre en compte les évolutions d'environnements dynamiques (c'est-à-dire en constante évolution) surveillés grâce à un certain nombre de données issues de mesures effectuées par des capteurs. Dans ces systèmes, une action réactive correspond à la définition des actions que le système doit effectuer en réponse à l'apparition de tel ou tel événement au cours de la résolution. Ces actions sont modélisées par des tâches auxquelles sont associées une ou plusieurs méthodes. Au cours de l'exécution d'une tâche, c'est le gestionnaire de tâches qui a pour rôle de sélectionner la méthode la plus adaptée au contexte parmi les méthodes concurrentes associées à celle-ci. Là encore, une méthode renvoie soit directement à l'exécution d'une action (on qualifiera alors la tâche correspondante de tâche ***élémentaire***), soit à une décomposition en sous-tâches (il s'agira alors d'une tâche ***composite***).

L'exemple présenté dans [Chaillot 93] est celui d'une bibliothèque de programmes de calcul matriciel (programmes extraits des bibliothèques de programmes mathématiques référencées par le ***Numerical Algorithms Group*** Ltd) . A la tâche d'***inversion de matrices*** sont associées différentes méthodes plus ou moins adaptées à la matrice à inverser en fonction de ses caractéristiques.

Le mécanisme de sélection dynamique de méthodes proposé dans ce travail repose sur une sélection hiérarchique (basée sur une classification hiérarchique des méthodes) combinée à une gestion des possibilités d'activation.

D. OCAPI

OCAPI (Outil de Contrôle Automatique de Procédures Images), [Moisan 97] [Clément 90] est un moteur de pilotage développé au sein du projet ***ORION*** de l'INRIA. ***OCAPI*** est le moteur de pilotage qui a servi notamment de noyau à ***PROGAL***, un système expert en traitement d'images de galaxies.

OCAPI, initialement développé pour piloter une bibliothèque de programmes de traitement d'images, a également été utilisé pour d'autres applications, démontrant ainsi que ***OCAPI*** permet de piloter des bibliothèques de programmes indépendamment du domaine abordé par les programmes qui constitue la bibliothèque.

Dans cette approche, un ***but*** représente une fonctionnalité du domaine concerné et un ***opérateur*** contient une connaissance spécifique pour résoudre un ***but*** donné, c'est-à-dire réaliser une tâche. Un ***but*** contient également la connaissance qui permet de choisir entre les différents ***opérateurs*** capable de le résoudre. Cette connaissance est exprimée par le biais de règles de choix qui tiennent compte des données du problème. Un opérateur peut-être ***complexe*** ou ***élémentaire*** selon qu'il nécessite ou non d'être décomposé en plusieurs sous-étapes. Un ***opérateur complexe*** renvoie donc à une décomposition en sous-étapes constituées d'***opérateurs***, tandis qu'un ***opérateur élémentaire*** renvoie directement à un programme exécutable.

La demande de résolution d'un problème s'exprime sous la forme d'une ***requête*** qui indique explicitement le but à atteindre. Outre le but à atteindre, une requête permet d'exprimer les contraintes sur les données d'entrée et de sortie du problème. Ces contraintes serviront ensuite pour choisir l'opérateur adéquate pour la résolution du problème abordé.

Deux systèmes ont été développé dans la lignée d'***OCAPI*** : ***PEGASE*** et ***PHENIX*** [Moisan 97].

IV. Les modèles hybrides

Dans cette partie, je vais présenter des modèles qui sont à mi-chemin entre les modèles conceptuels et les modèles opérationnels : leur objectif est de fournir un formalisme de représentation des connaissances adaptée à la conception – i.e. à la modélisation – de SBC qui puisse permettre une opérationnalisation – i.e. implémenation – automatique ou semi-automatique du modèle établi durant la phase de conception.

A. LISA

LISA [Delouis 93] est un langage de description dédié à la conception de **SBC** qui permet de passer d'un modèle conceptuel à un modèle opérationnel. Dans ***LISA***, un ***but*** caractérise un problème à résoudre et chaque but est associé aux méthodes les plus adaptées à son obtention. Une méthode caractérise donc un savoir-faire en décrivant la stratégie de résolution permettant d'atteindre le but recherché ; celle-ci peut être l'exécution directe d'un programme externe, l'utilisation d'une règle d'inférence sur la base de connaissance ou une décomposition en sous-buts. Le choix de la méthode à utiliser pour atteindre un but donné parmi un ensemble de méthodes concurrentes est guidé par la base de règles de préférence associé à ce but.

Pour décrire un ***but***, respectivement une ***méthode***, l'utilisateur du langage ***LISA*** dispose des fonctions **DEFBUT**, respectivement **DEFMETHOD**, présentées sur les figures suivantes. Les attributs **METHODES** et **PREFERENCES** de la fonction **DEFBUT** permettent de spécifier les méthodes associées au but considéré ainsi que les règles de préférence qui permettent de choisir la méthode la mieux adaptée au contexte. L'attribut **STRATEGIE** de la fonction **DEFMETHOD** permet, quant à lui, de définir la stratégie de résolution permettant d'atteindre le but cible ; il peut s'agir d'une procédure de calcul (i.e. un programme), d'un ensemble de règles d'inférence à utiliser sur la base de connaissances du **SBC**, ou d'une décomposition en sous-buts.

```
DEFBUT        <nom-but>
   (TITRE                  <identificateur>)
   (DESCRIPTION            <description>)
   (ETAT-BUT               <état-but>)
   (CONTEXTE               <contexte>)
   (METHODES               <nom-méthode>*)
   (PREFERENCES            <préférences>)
```

Fig. 10 : Fonction ***LISA*** permettant de définir un ***but***

```
DEFMETHOD  <nom-méthode>
   (TITRE                  <identificateur>)
   (DESCRIPTION            <description>)
   (PARAMETRES             <paramètres>)
   (STRATEGIE              <nom-procédure> |
                           <règles-d'inférence> |
                           <décomposition-en-sous-buts>)
```

Fig. 11 : Fonction ***LISA*** permettant de définir une ***méthode***

B. KSM

KSM (***Knowledge Structure Manager***) est un outil qui permet de développer un modèle conceptuel et de construire de façon automatique une version opérationnelle de celui-ci. Pour la création d'un SBC, ***KSM*** s'appuie sur le concept d'***unité de connaissances*** ; celles-ci permettent de décrire toutes les connaissances relatives à un domaine ainsi que les fonctions

qui y sont associées. Les connaissances d'un domaine sont récursivement réparties sur plusieurs sous-domaines par le biais de nouvelles unités de connaissances ; les unités de connaissances obtenues à la fin de cette « décomposition » qui ne sont donc pas décomposées en sous-domaines sont dénommées ***unités primaires de connaissances***.

Dans ce formalisme, une fonctionnalité associée à une unité de connaissances est représentée par une tâche. Une telle tâche reçoit en entrée un ensemble de données qu'elle utilise pour produire un ensemble de données en sortie. L'exécution d'une tâche est alors effectuée par le biais des méthodes associées. Ainsi, une tâche associée à une unité de connaissances est récursivement décomposée en sous-tâches par le biais de ses méthodes associées. Dans cette représentation, une méthode indivisible représente donc une fonctionnalité d'une unité primaire de connaissances.

L'opérationnalisation du modèle conceptuel est réalisée par le biais du langage ***LINK*** qui permet de décrire une procédure d'inférence - appelée ***primitive de représentation*** – pour chaque unité primaire de connaissances du modèle conceptuel. Par ce biais, ***KSM***, qui gère une bibliothèque de ***primitives de représentation*** dont chaque élément est associé à un module logiciel réutilisable, est capable d'associer à chaque fonctionnalité des unités primaires de connaissances un mécanisme opérationnel.

C. PROTEGE

PROTÈGÈ-I [Musen 89, Musen 93] est un programme de méta-niveau qui génère des outils d'acquisition des connaissances adaptées à des classes d'applications : on parlera de tâches opérationnelles. ***PROTÈGÈ-I*** inclut un modèle de résolution de problèmes : le raffinement épisodique de squelettes de plans (***Episodic Skeletal Plan Refinement*** ou ***ESPR***). Les cogniticiens étendent ce modèle de résolution de problèmes avec des connaissances du domaine afin de définir des modèles spécifiques aux domaines d'applications abordés.

PROTÈGÈ-I est donc un outil d'acquisition des connaissances, orienté méthodes (c'est-à-dire construit autour d'une méthode de résolution de problèmes) qui demande à ses utilisateurs de définir les connaissances du point de vue de la méthode sous-jacente (le raffinement épisodique de squelettes de plans). Contrairement à d'autres outils d'acquisition des connaissances, ***PROTÈGÈ-I*** produit en sortie, non pas une base de connaissance, mais un autre outil d'acquisition des connaissances orienté tâches (c'est-à-dire instanciant le modèle de résolution de problèmes pour pouvoir aborder une classe d'applications par le biais de tâches opérationnelles), ce qui fait de ***PROTÈGÈ-I*** un méta-outil. Les cogniticiens définissent via ***PROTÈGÈ-I*** les structures de tâches relatives à une classe d'applications qui peuvent être résolues en utilisant la méthode ***ESPR***; ***PROTÈGÈ-I*** produit alors un autre

S.C.D. - U.H.P. NANCY 1 SCIENCES BIBLIOTHÈQUE

outil d'acquisition des connaissances adapté à la classe d'applications en question. C'est alors aux spécialistes du domaine qu'incombe la tâche d'utiliser ce second outil afin de définir l'ensemble des connaissances spécialisées relative à telle ou telle application en particulier. L'outil d'acquisition orienté tâches produit alors une base de connaissances qui peut directement être utilisée par un moteur d'inférence générique (**e-ONCOCIN**).

Dans ***PROTÈGÈ-II***, le but a été d'étendre ***PROTÈGÈ-I*** de façon à permettre l'acquisition de connaissances non seulement du point de vue d'une seule et unique méthode de résolution de problèmes, de surcroît prédéfinie (l'***ESPR***), mais aussi du point de vue d'un ensemble de méthodes paramétrables qui pourront aborder des problèmes de granularités différentes. Ainsi, un des enjeux importants pour ***PROTÈGÈ-II*** est de faciliter non seulement la réutilisation des méthodes de résolution de problèmes comme le raffinement épisodique de squelettes de plans, mais aussi celle des ontologies qui décrivent les concepts des différents domaines qui pourront être abordés par ces différentes méthodes de résolution de problèmes.

Dans ***PROTÈGÈ-II***, une tâche représente une activité du monde réel (ou plutôt son abstraction) ; elle produit un certain nombre de données en sorties à partir des données qui lui ont été soumises en entrées. Une tâche ne spécifie pas directement la transformation qui lui permet de produire ses sorties à partir de ses entrées, elle le fait par le biais de sa méthode associée. De façon informelle, une méthode est une procédure de transformation partiellement prédéfinie qui transforme l'ensemble de ses entrées en un ensemble de sorties suivant un ensemble de règles qui sont exprimé à un niveau abstrait (au ***knowledge level***). Une méthode réalise la tâche de transformation pour laquelle elle a été prévue en faisant appel à un certain nombre de sous-tâches. Chacune de ces sous-tâches est alors réalisée soit par le biais d'une autre méthode (avec ses propres sous-tâches), soit par une procédure primitive de résolution de problème qui ne nécessite pas de décomposition en sous-tâches. Ces procédures primitives qui ne sont pas décomposables sont appelées mécanismes (***mecanisms***). Ainsi, une méthode de résolution de problèmes n'est que partiellement prédéfinie puisque le détail des algorithmes utilisés pour générer les sorties de celle-ci à partir de ses entrées dépend des sous-méthodes et des mécanismes qu'elle utilisera pour réaliser ses sous-tâches.

L'un des principaux objectifs de ce travail étant la réutilisation des connaissances et des méthodes de résolution de problèmes associés, ***PROTÈGÈ-II*** permet la construction de ***bibliothèques*** de mécanismes qui contient non seulement les mécanismes eux-mêmes, mais aussi la description des méthodes et des tâches et les ontologies des différents domaines abordables.

V. Conclusions

A. Clarification du vocabulaire

Nous avons présenté un certain nombre de systèmes d'aide à la résolution de problèmes avec leur modèle de tâches. Le lecteur aura sans doute pu remarquer qu'il existe presque autant de formulations que de systèmes de résolution de problèmes basés sur des descriptions des processus de résolution par modèles de tâches. Cependant, il est possible d'établir une correspondance entre les expressions utilisées au sein de ces modèles et celles que nous utiliserons dans ce mémoire et que nous avons tenté de définir dans le lexique qui se trouve au début de ce document. Le tableau ci-dessous a pour but de réaliser cette correspondance :

Concept Abordé	Notre Terminologie	Au niveau de l'Etat de l'Art Présenté	
		Système/Approche Concernée	Terminologie Correspondante
Représentation d'un problème	***Tâche***	**OCAPI, LISA**	***But***
Décomposition successive d'un problème en sous-problèmes	***Stratégie***	**Tâches Génériques de CHANDRASEKARAN**	Décomposition d'une méthode en sous-tâches
		OCAPI	Corps d'un opérateur complexe
		CommonKADS, PowerTasks	***Task-body***
		TASK	***Corps***
Résolution d'un problème simple	***Tâches Elémentaires***	**TRAM**	Programmes
		OCAPI	***Opérateurs élémentaires***
		KSM	***Primitives de représentation***
		PROTÉGÉ-II	***Mécanismes***
		SCARP	***Procédures***
		CommonKADS	***Primitives*** ou ***Tâches terminales***

Fig. 12 : Tableau de correspondance entre les termes utilisés dans les différents formalismes (seuls sont présentés les systèmes dont la terminologie diffère de la nôtre)

B. Décomposition Tâche/Méthode/Sous-Tâche Vs Tâche /Sous-Tâche

Dans les travaux présentés dans cet état de l'art, nous avons été amené à distinguer deux classes de systèmes suivant le type de décomposition adoptée – comme cela a déjà été mentionné au paragraphe B.3. de l'introduction du présent chapitre – ; en effet, les modèles de tâches utilisés reposent tantôt sur une décomposition de type Tâche/Méthode/Sous-Tâche, tantôt sur une décomposition de type Tâche/Sous-Tâche.

En fait, cette différence de représentation dans la décomposition d'un problème en sous-problèmes plus simple repose plus directement sur le nombre d'entités impliquées dans ces décomposition. Ainsi, dans les modèles admettant une décomposition de type Tâche/Méthode/Sous-Tâche, le nombre d'entités différentes est de 2, tandis que pour ceux ayant retenu une décomposition de type Tâche/Sous-Tâche, il n'y a qu'un seul type d'entité.

Ainsi, un modèle comme OCAPI, qui n'admet pas une décomposition de type Tâche/Méthode/Sous-Tâche, mais du type But/Requête/Opérateur, est lui-aussi basé sur 2 types d'éléments : les ***Requêtes*** et les ***Opérateurs*** et est donc à ce titre à rapprocher de cette classe de méthodes dont la décomposition repose sur la description d'une alternance d'objets de 2 types.

Sur le fond, signalons que le fait d'utiliser un formalisme basé sur 2 types d'objets différenciés (que ce soit les ***Tâches*** et les ***Méthodes*** ou les ***Requêtes*** et les ***Opérateurs***) sur un seul type d'objet (les ***Tâches***) ne change en rien le pouvoir d'expression des modèles présentés. Ce choix me semble être lié, le plus souvent, à des choix d'ordre implémentatoires ou d'ordre « philosophique » - il est vrai que distinguer l'action de la manière de la réaliser peut sembler être une bonne solution sur le plan de la sémantique (cela permet de mettre l'accent sur l'aspect fonctionnel), cependant, dans le cadre d'un travail opérationnel, nous ne considérons, le plus souvent, que les actions qui sont associées plus ou moins directement à des réalisations opérationnelles ce qui peut justifier le fait qu'un action est indissociable de la façon de la réaliser -.

Système/Approche	Nbre d'Entités Intervenant dans la Décomposition	Formalisme de Décomposition
CommonKADS	1	Tâche/Sous-Tâche
KADS	1	Tâche/Sous-Tâche
TASK	1	Tâche/Sous-Tâche
PowerTasks	1	Tâche/Sous-Tâche
SCARP	1	Tâche/Sous-Tâche
OCAPI	2	But/Opérateur/Requête

Tâches Génériques de CHANDRASEKARAN	2	Tâche/Méthode/Sous-Tâche
KSM	2	Tâche/Méthode/Sous-Tâche
LISA	2	Tâche/Méthode/Sous-Tâche
PROTÉGÉ-I &II	2	Tâche/Méthode/Sous-Tâche
TRAM	2	Tâche/Méthode/Sous-Tâche

Fig. 13 : Tableau de synthèse résumant les formalismes de décomposition retenus

C. Approches Conceptuelles Vs Opérationnelles

Dans ce chapitre, nous avons également distingué 3 types de modèles en fonction de leur vocation à être des outils de conceptualisation (aide à formalisation et à la représentation de connaissances abstraites) ou d'opérationnalisation (représentation et manipulation de savoir-faire applicatifs).

Système/Approche	Type
CommonKADS	Conceptuel
KADS	Conceptuel
TASK	Conceptuel
Tâches Génériques de CHANDRASEKARAN	Conceptuel
KSM	Hybrides
LISA	Hybrides
PROTÉGÉ-I &II	Hybrides
PowerTasks	Opérationnel
SCARP	Opérationnel
OCAPI	Opérationnel
TRAM	Opérationnel

Fig. 14 : Tableau de synthèse rappelant les vocations des différents modèles présentés

D. Synthèse

Le tableau suivant tente de présenter de façon concise les différents systèmes présentés dans ce chapitre afin d'en donner une vue d'ensemble.

Système/Approche	Type	Nbre d'Entités Intervenant dans la Décomposition	Formalisme de Décomposition	Domaine d'Application
CommonKADS	Conceptuel	1	Tâche/Sous-Tâche	Générique
KADS	Conceptuel	1	Tâche/Sous-Tâche	Générique
TASK	Conceptuel	1	Tâche/Sous-Tâche	Conception de ***SBC***
Tâches Génériques de CHANDRASEKARAN	Conceptuel	2	Tâche/Méthode/Sous-Tâche	Conception de ***SBC***
KSM	Hybrides	2	Tâche/Méthode/Sous-Tâche	Conception de ***SBC***
LISA	Hybrides	2	Tâche/Méthode/Sous-Tâche	Conception de ***SBC***
PROTÉGÉ-I &II	Hybrides	2	Tâche/Méthode/Sous-Tâche	Conception de ***SBC***
PowerTasks	Opérationnel	1	Tâche/Sous-Tâche	Générique
SCARP	Opérationnel	1	Tâche/Sous-Tâche	Générique
OCAPI	Opérationnel	2	But/Opérateur/Requête	Traitement d'Images
TRAM	Opérationnel	2	Tâche/Méthode/Sous-Tâche	Générique

Fig. 15 : Tableau de synthèse présentant les différents modèles abordés dans ce chapitre

Chapitre 2

GALAXIE

Un Environnement d'Aide à la Conception et au Maquettage d'Applications de Traitement du Signal et d'Interprétation de Signaux basé sur une Bibliothèque de Programmes « Intelligente »

I. Notre Modèle de Tâches

A. Introductions et généralités

A.1. Positionnement

Si l'on reprend le tableau de synthèse précédent et que l'on tente de situer notre réalisation dans ce cadre, nous obtenons le tableau suivant :

Système/Approche	Type	Nbre d'Entités Intervenant dans la Décomposition	Formalisme de Décomposition
CommonKADS	Conceptuel	1	Tâche/Sous-Tâche
KADS	Conceptuel	1	Tâche/Sous-Tâche
TASK	Conceptuel	1	Tâche/Sous-Tâche
Tâches Génériques de CHANDRASEKARAN	Conceptuel	2	Tâche/Méthode/Sous-Tâche
GALAXIE	**Hybrides**	**1 ou 3**	**Tâche/Sous-Tâche ou Requête/Méta-Tâche/Tâche/Sous-Tâche**
KSM	Hybrides	2	Tâche/Méthode/Sous-Tâche
LISA	Hybrides	2	Tâche/Méthode/Sous-Tâche
PROTÉGÉ-I &II	Hybrides	2	Tâche/Méthode/Sous-Tâche
PowerTasks	Opérationnel	1	Tâche/Sous-Tâche
SCARP	Opérationnel	1	Tâche/Sous-Tâche
OCAPI	Opérationnel	2	But/Opérateur/Requête
TRAM	Opérationnel	2	Tâche/Méthode/Sous-Tâche

Fig. 16 : Tableau de synthèse présentant le positionnement de notre système ***GALAXIE***

Notre travail – et en conséquence, notre système – c'est essentiellement inspiré des travaux réalisés autour des ***Tâches Génériques*** de Chandrasekaran, de ***PROTÈGÈ***, de ***SCARP*** et d'***OCAPI***.

Cependant, nous avons essayé d'apporter notre contribution apportant un certain nombre d'améliorations à notre modèle par rapport aux modèles dont nous nous sommes plus directement inspiré (***OCAPI***, ***SCARP***) :

- introduction des requêtes et extension de leur utilisation de sorte à permettre de spécifier une stratégie de résolution non pas par sa ou ses méthodes associées, mais par la définition des différents sous buts intervenant dans la résolution (possibilité d'indéterminisme dans les stratégies). La première raison de cet ajout est de permettre la mise en place d'une architecture ouverte client-serveur dans laquelle l'utilisateur (ou le système) pose un problème par le biais d'une requête et pour lequel il obtient une solution. Le second intérêt de cet ajout est de permettre à l'utilisateur de manipuler des tâches qu'il sait être ré solvables, mais sont il ignore les moyens de résolution associés ;

- extension de l'expressivité de la grammaire de description des stratégies en permettant aux opérateurs de porter non seulement sur des éléments de bases (les tâches), mais aussi sur des éléments complexes : les stratégies elles-mêmes. L'intérêt de ce choix est d'offrir un langage à la fois souple et puissant à l'utilisateur pour qu'il puisse exprimer ses connaissances et ce avec la granularité de description qui lui semblera le plus adapté à ses besoins ;

- extension de notre modèle de tâche initial et distribution du mécanisme de pilotage associé de sorte à permettre une utilisation qui puisse s'intégrer dans un paradigme multi-agent. Les apports de cette extension – qui constitue la pierre angulaire de notre travail – sont multiples : tout d'abord, cela doit permettre de réintroduire de l'indéterminisme et de l'opportunisme dans les mécanismes de construction de solution, car le plus souvent, dans le cadre de problème complexe, on ne peut se contenter de dérouler des stratégie pré-établies. Cela pourra aussi permettre de mettre en place différentes stratégies de coopération entre les entités résolvantes ; enfin cela peut permettre la distribution géographique des connaissances et des mécanismes de résolution.

A.2. Remarques préliminaires

Cette section décrit le modèle de tâche issu de notre travail. Une première version de ce modèle a été mis en application pour traiter le problème de la séparation de sources. Rappelons à cette occasion que la séparation de sources vise à retrouver, le plus précisément possible, à partir des signaux observés – qui sont, en fait, la superposition des signaux émis par différentes sources – les sources qui sont à l'origine de la manifestation observée que l'on cherche à interpréter.

Bien que cette application valide partiellement notre approche, elle a été l'occasion d'un certain nombre de réflexions qui nous ont poussés à étendre notre modèle initial. Une application plus conséquente mériterait donc d'être développée afin de mettre en œuvre et de tester ce modèle étendu qui, pour l'instant, n'a pas encore été utilisé.

Notre modèle de tâches permet de décrire des problèmes - un problème est ici une tâche ou action complexe qui doit être réalisée - et les moyens permettant de les résoudre : nous supposons donc au préalable qu'à chaque problème correspond une tâche qui le décrit et qui explicite une façon de le résoudre.

Un problème - ou la tâche associée - peut alors être décrit, du point de vue externe, par ce sur quoi il porte et par les résultats que sa résolution fournit, c'est-à-dire par ses entrées et ses sorties. D'un point de vue interne, il faut pouvoir décrire la stratégie de résolution associée, ainsi que tout ce qui est nécessaire à la bonne exécution de cette même stratégie. Il est alors possible de scinder la description d'un problème - et donc d'une tâche, puisque pour nous un problème est décrit par une tâche - en deux grands blocs distincts, l'un rattaché à la description du problème à résoudre - il s'agit alors du contexte d'application de la tâche - et l'autre relatif à sa résolution - c'est-à-dire décrivant l'exécution proprement dite de la tâche -.

Par la suite, nous commencerons par nous intéresser au contexte d'application d'une tâche (quand et comment l'utiliser), puis nous nous focaliserons sur la description de son exécution.

A.3. Les différents types de connaissances nécessaires au fonctionnement de notre "Programmathèque"

Pour nos besoins, nous avons défini au sein de notre modèle de résolution de problèmes différents types de connaissances : les requêtes, les méta-tâches, les tâches et les entités :

- les ***requêtes*** permettent à l'utilisateur (et/ou au système, au cours de la résolution) de poser le problème qu'il souhaite voir résoudre : pour ce faire, il précise à la fois l'action qu'il souhaite voir se réaliser et les données sur lesquelles il souhaite que cette dernière soit effectuée ;

S.C.D. - U.H.P. NANCY 1 SCIENCES BIBLIOTHÈQUE

- les ***méta-tâches*** qui viennent chapeauter un ensemble de tâches réalisant la même action (i.e. permettant d'atteindre le même but), action spécifiée par la requête en cours de traitement ; les méta-tâches permettent d'associer une action/but à un ensemble de tâches et de spécifier des règles de choix et d'évaluation des différentes tâches concurrentes ;

- les ***tâches*** permettent de modéliser un problème existant – i.e. une action que l'on désire réaliser, sans pour autant savoir précisément comment y parvenir - ; une tâche renvoie soit directement à un algorithme permettant de résoudre le problème posé (tâche simple ou primitive), soit à une stratégie de résolution qui décrit comment la tâche peut être décomposée en sous-tâches (ou sous-requêtes) plus élémentaires (tâche complexe) ; une tâche renvoie donc finalement à un ensemble d'algorithmes permettant d'atteindre le but fixé : réaliser une action donnée ;

- les ***entités :*** chaque ***entité*** définit un objet manipulé dans le domaine abordé par le système de résolution de problème (et permet ainsi son encapsulation).

Remarques :

1. Les méta-tâches n'interviennent que pour permettre au système d'effectuer des choix implicites entre plusieurs tâches permettant de résoudre le problème cible ; ainsi, dans le cadre de la séparation de source, le choix de la méthode à appliquer (les méthodes candidates, Sobi, Cardoso, PFS, Hérault-Jutten, sont présentées succinctement dans les chapitre 3) est explicite, ce qui justifie le fait que nous n'ayons pas utilisé cette structure dans notre maquette ;

2. L'utilité de définir des entités est apparue au cours de la réalisation de notre maquette. Ainsi dans celle-ci, les matrices de signaux manipulées n'ont pas été encapsulées dans des entités : les tâches travaillent directement sur les objets qui sont stockés dans une zone de travail partagée par le biais de fichiers ; De même, les structures de requêtes sont apparues à la suite de l'évaluation que nous avons faite de notre maquette ;

3. Les règles de choix et d'évaluation associées aux tâches permettent, dans le cadre d'un choix système de préciser les règles permettant de choisir une tâche parmi un ensemble de tâches et d'en évaluer les résultats. Les règles d'adaptations sont là pour permettre l'initialisation et la mise à jour des éventuels paramètres d'une tâche. Ces différentes structures sont apparues

nécessaires pour permettre à notre moteur de pilotage d'effectuer des choix « plus intelligents ».

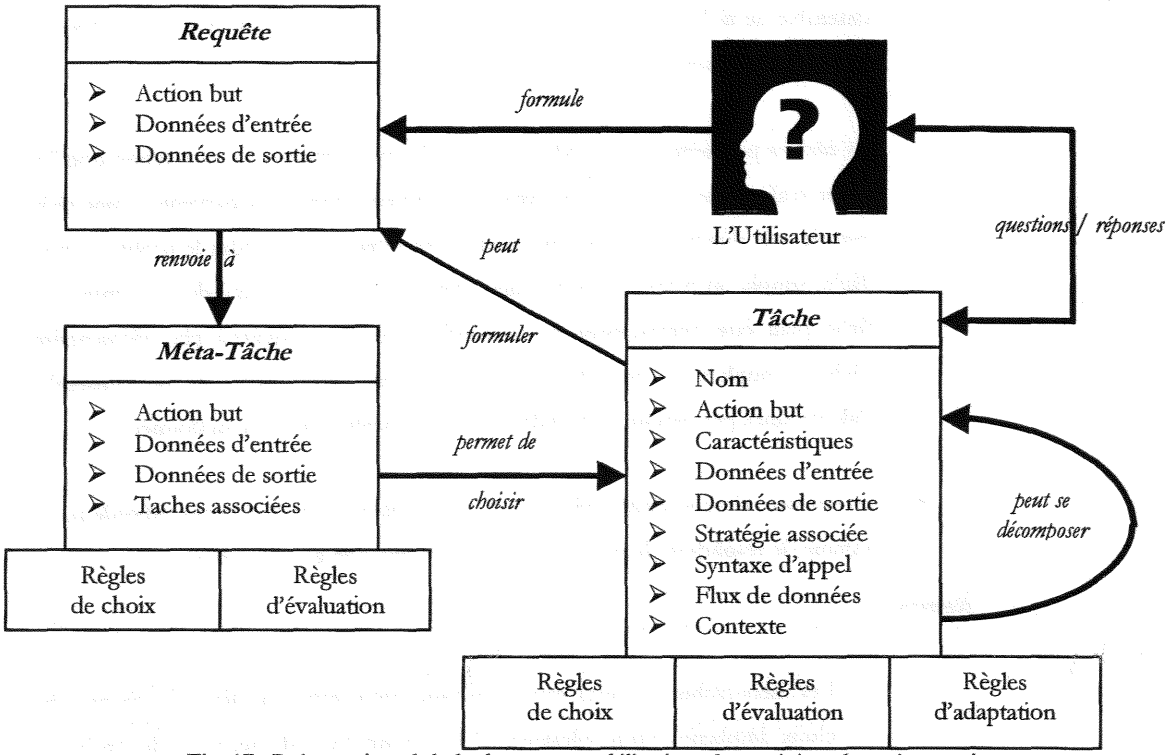


Fig. 17 : Présentation globale de notre modélisation : les entités et leurs interactions

A.4. Quelques définitions

Avant d'aller plus loin dans la présentation de notre modèle de tâches, il convient de préciser un certain nombre de termes et de notions sur lesquels il est indispensable de s'accorder.

Comme nous l'avons déjà signalé, pour permettre à notre bibliothèque de programmes intelligente de choisir le meilleur moyen de réaliser une tâche cible, il est nécessaire de modéliser ce que nous appellerons son contexte d'application. Un « contexte » est la structure de contrôle - qui représente le point de vue adopté pour réaliser une tâche complexe et qui reflète ainsi l'état du monde dans lequel on doit réaliser cette tâche – qui permet l'adaptation d'une tâche à l'environnement - ou contexte - dans lequel elle doit être réalisée ; ainsi pour savoir quelle méthode utiliser lorsque nous cherchons à séparer des sources, il convient tout d'abord de déterminer les caractéristiques de la matrice des observations (nous reviendrons plus tard sur ce problème). La sélection d'un contexte résulte donc de l'identification d'une situation spécifique.

Lorsque l'on s'intéresse à la réalisation d'une action complexe, on est amené à distinguer 2 types de tâches :

- les ***tâches complexes*** (ou ***composées***) : elles sont résolues par des plans d'actions - c'est-à-dire des séquences initialement prédéfinies, mais pouvant être dynamiquement adaptées, de tâches élémentaires - obtenus grâce à un planificateur. Ainsi, résoudre une tâche complexe revient à parcourir son arbre de décomposition/spécialisation ;

- les ***tâches primitives*** (ou ***élémentaires***) : elles correspondent, en fait, à des tâches «simples » directement associées à des algorithmes ou a des exécutables ; elles correspondent aux feuilles des arbres de décomposition/spécialisation des tâches complexes.

Une tâche peut donc être soit directement associée à une méthode directe (un module algorithmique ou un exécutable), soit associée à une stratégie de résolution permettant sa réalisation par le biais d'une décomposition/spécialisation. Pour une action donnée, le système peut alors disposer à la fois d'une méthode directe (tâche primitive) ou d'une stratégie de résolution par décomposition/spécialisation (tâche complexe) ; le rôle du moteur de pilotage de programmes de notre bibliothèque intelligente est alors de choisir dynamiquement - en fonction du contexte d'application de la tâche et essentiellement du contexte d'exécution - parmi l'ensemble des tâches disponibles, celle qui doit être appliquée. Pour ce faire, le système fera appel aux règles de choix associées à l'action en question.

A.5. Un exemple de tâche : la séparation de source

Signalons au préalable que le but de l'application que nous avons développé est d'aider un utilisateur non expert à utiliser au mieux les méthodes de séparation de sources qui sont à sa disposition, en fonction des caractéristiques intrinsèques de son problème, mais aussi en fonction des résultats observés après utilisation de telle ou telle méthode déjà essayée.

Nous allons maintenant présenter l'arbre de décomposition/spécialisation de la tâche de séparation de sources. La formalisation présentée ici correspond à celle exposée dans le document intitulé « Qu'est-ce que la séparation de sources ? ». Signalons au passage que celle-ci diffère quelque peu de celle implémentée dans notre maquette du fait que les tâches de capture, de conditionnement et de synchronisation des signaux sont supposées réalisées par le matériel.

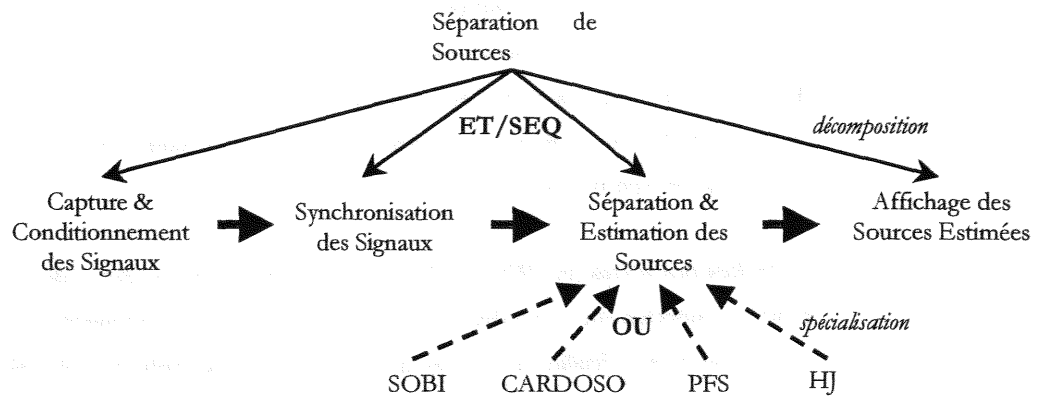


Fig. 18 : Graphe de décomposition/spécialisation de la séparation de sources

Le système sait donc que pour réaliser la tâche ***Séparation de Sources***, il devra séquentiellement effectuer les tâches de ***Capture & Conditionnement des Signaux***, de ***Synchronisation des Signaux***, de ***Séparation & Estimation des Sources*** et d'***Affichage des Sources Estimées***. Par contre, lorsqu'il cherchera à effectuer la tâche ***Séparation & Estimation des Sources***, il devra spécialiser celle-ci en effectuant un choix entre les différentes méthodes qui lui sont proposées ; ce choix sera contexte-dépendant et sera guidé par les règles de choix qui lui sont associées.

Avec cet exemple, on s'aperçoit de la nécessité de définir un certain nombre d'opérateurs permettant la description de la stratégie de résolution associée à la tâche complexe que l'on cherche à effectuer. Ainsi, dans l'exemple précédent, on a besoin des 2 opérateurs suivants :

- le OU, opérateur de spécialisation, permettant de spécifier une alternative qui est ici représenté par un choix système entre plusieurs algorithmes ;

- le ET SEQUENTIEL permettant de préciser que l'exécution de tâches doit être effectuer en séquence.

Sur l'arbre développé, on peut alors distinguer deux types de nœuds :

- les nœuds internes qui sont associés à des tâches nécessitant une décomposition en sous-tâches - car non élémentaires - ou nécessitant d'être spécialisées.

- les feuilles qui sont nécessairement associées à une méthode directe permettant de réaliser la sous-tâche correspondante : elles correspondent à des tâches non spécialisables et non décomposables, suffisamment précises pour admettre des méthodes associées.

Enfin, cet exemple est intéressant car il fait apparaître la nécessité de « préciser » - on utilisera le terme ***« spécialiser »*** - certaines caractéristiques d'une tâche générique - comme par exemple la tâche ***« Séparation & Estimation des Sources »*** avant de pouvoir exécuter la tâche but - ici, « séparer des sources » -. Ainsi, on peut être amené à définir une relation de ***spécialisation*** sur l'ensemble des tâches qui induit une organisation hiérarchique sur celui-ci ; on dit alors que la tâche ***T'*** *est plus spécifique que la tâche* ***T*** si le domaine des valeurs des paramètres de ***T'*** est plus réduit que celui de ***T*** et si ***T'*** et ***T*** résolve le même problème.

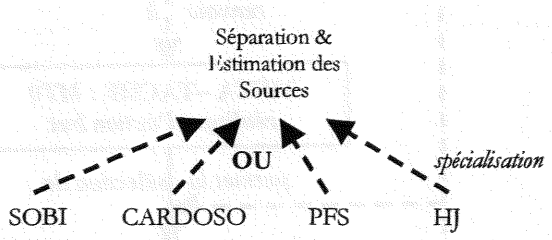


Fig. 19 : La spécialisation de la tâche générique ***Séparation de Sources***

Ainsi, ***SOBI***, ***CARDOSO***, ***PFS*** et ***HJ*** sont autant de spécialisations de la tâche ***« Séparation & Estimation des Sources »***. Notons au passage, que cette dernière est une tâche complexe dont la stratégie associée est un choix explicite entre les différentes méthodes qui la spécialise.

A.6. Pilotage de programmes et planification

Une tâche complexe pouvant être - dans le cadre de notre modélisation - décomposée récursivement en sous-tâches de plus en plus élémentaires, le moteur de pilotage de programmes de notre bibliothèque intelligente doit pouvoir construire l'enchaînement de « tâches primitives » - ou méthodes directes - le mieux adapté au problème abordé. Pour ce faire - et du fait du formalisme retenu pour la description des tâches -, on est amené à utiliser un mécanisme de planification hiérarchique - planification dans un espace de plans - alternant des phases de spécialisation et de décomposition : il s'agit, en fait, d'explorer l'arbre de décomposition/spécialisation associé à la tâche résolvant le problème posé.

Ainsi, une fois la méta-tâche cible identifiée, ***MT0***, – grâce à l'action qu'elle réalise et aux types de ses données d'entrée et de sortie qui sont celles spécifiées dans la requête ***R0*** -, le système va choisir la tâche associée la mieux adaptée, ***T0***, et développer sa stratégie de résolution associée, décomposant ainsi la tâche initiale en sous-tâches plus simples (***ST1***, ***ST2***, ***ST3***, etc.). Ensuite, le système va procéder de même - et de façon récursive - avec tous les sous-tâches obtenus (***ST1.1***, ***ST1.2***, ***ST1.3***, etc.) : il va ainsi considérer les unes après les autres les stratégies associées à ces différentes sous-tâches, décomposant ainsi chacune d'entre elles

en sous-tâches de plus en plus élémentaires, résolues récursivement jusqu'à obtention de tâches « simples » qui se résolvent directement par exécution d'un programme.

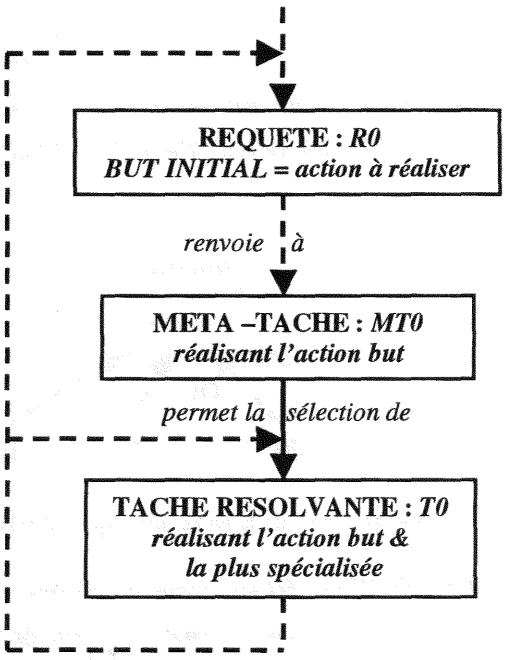


Fig. 20 : Initialisation du processus de résolution de problème

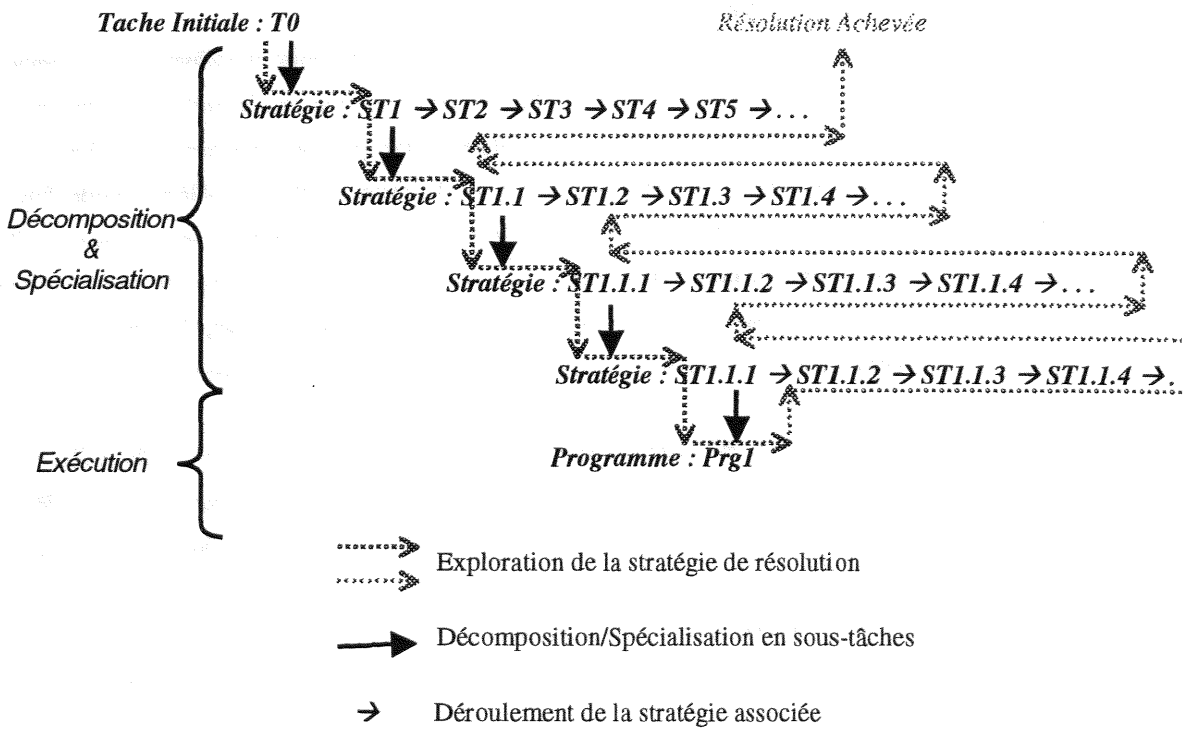


Fig. 21 : Processus récursif de résolution de problème par décomposition/spécialisation

Les plans successifs sont donc construits avec des tâches de plus en plus spécialisées. Cela revient à exécuter une succession de phases de spécialisation et de décomposition afin d'obtenir finalement un plan ne contenant que des méthodes ; les phases de spécialisation permettent de détecter la sous-tâche spécialisée la plus adaptée aux variables d'entrée du problème - et à son contexte - et les phases de décomposition servent à décomposer la tâche en sous-tâches pour lesquelles on adoptera la même stratégie (de décomposition) si elles ne sont pas élémentaires. Signalons à cette occasion que les phases de spécialisation correspondent ici aux choix explicitement invoqués par l'utilisateur au sein des différentes stratégies qu'il a décrit.

Remarques : La ***« spécialisation »*** de tâche est l'action qui consiste, à partir d'une tâche générique, à choisir la tâche la plus adaptée au problème abordé, i.e. la plus spécifique. Contrairement à ce qui est fait dans les formalismes objet - où la spécialisation est directement calquée sur le mécanisme d'héritage de classes (c.f. ***SCARP*** & ***AROME***) - dans le cadre de notre modèle, la spécialisation consiste, en fait, en un choix dépendant uniquement du type des entrées/sorties de la requête à résoudre et de la tâche qui doit y parvenir.

A.7. Un exemple de tâche complexe : retour à la séparation de sources

La tâche complexe « Séparation de Sources » peut être caractérisée de la façon suivante :

- ses entrées : la matrice des observations ;
- ses sorties : la matrice des signaux séparés et la matrice de mélange ;
- ses méthodes associées : il existe différentes méthodes directes de séparation de sources et l'algorithme à utiliser pour la séparation dépend des caractéristiques de la matrice des observations. Les méthodes retenues par les experts de la **DRD** (***SOBI***, ***CARDOSO***, ***PFS***, ***Hérault-Jutten***) sont présentées dans le chapitre 3.

Remarque : L'analyse des experts de la **DRD** s'étant progressivement affinée, la tâche de « Séparation de Sources » implantée dans notre maquette diffère quelque peu de la version la plus récente présentée ci-avant. En conséquence, nous allons présenter ici le graphe de décomposition/spécialisation que nous avons utilisé dans notre maquette.

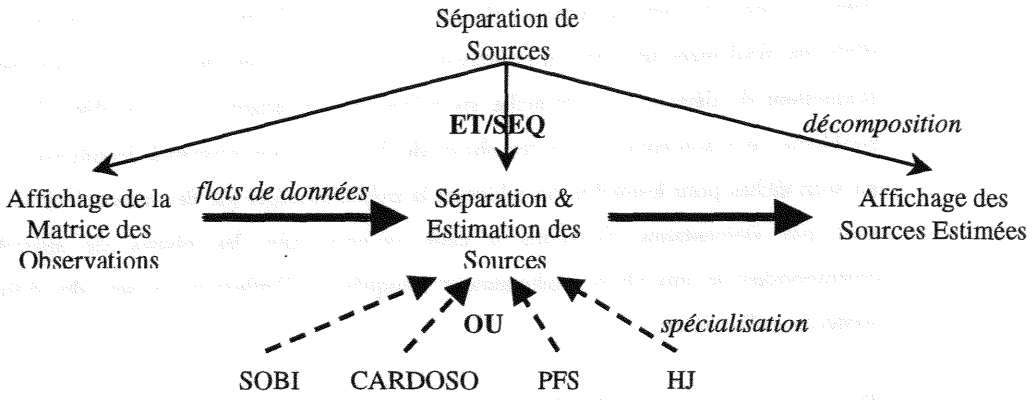


Fig. 22 : Graphe de décomposition/spécialisation utilisé dans notre maquette pour la séparation de sources

A.8. Nécessité de définir une grammaire de description des stratégies de résolution

A chaque tâche est associée une ***stratégie de résolution*** ; celle-ci permet de définir les différentes étapes de la résolution et le flot de contrôle qui ordonne ces étapes. Chaque étape de résolution est une tâche complexe ou primitive spécifiée par son nom ; le flot de contrôle est, quant à lui, décrit par le biais d'un certain nombre d'opérateurs permettant de combiner les tâches : séquence, parallèlisation, choix système, choix utilisateur, itération, tâche utilisateur. En effet, le flot de contrôle est défini par une combinaison de ces opérateurs suivant une grammaire que nous allons définir plus précisément.

L'opérateur de séquencement (***seq***) indique que toutes les étapes spécifiées en attribut doivent être résolues les unes après les autres. L'opérateur de parallèlisation (***par***) signifie que toutes les étapes énoncées doivent être exécutées, mais éventuellement en parallèle - ce qui signifie, en fait, qu'elles sont indépendantes en terme de flot de données -. Les opérateurs de choix (système ou utilisateur) permettent d'introduire un choix entre plusieurs étapes, soit sur l'initiative du système (***csy***), soit sur l'initiative de l'utilisateur (***cus***). Ce sont ces opérateurs de choix qui permettent de réaliser les phases de spécialisation de tâches qui rendent possible la prise en compte du contexte d'exécution de sorte à orienter le choix de la tâche spécialisée à effectuer en cours d'exécution d'une stratégie de résolution : les opérateurs de choix permettent ainsi, après exécution d'une partie de la stratégie, de choisir, en fonction du contexte ou des résultats déjà obtenus, la tâche la mieux adaptée.

```
<liste-de-stratégies>   :=   <stratégie> <liste-de-stratégies>   ||
                             <stratégie>

<stratégie>             :=   (séquence <liste-de-stratégies>)     ||
                             (parallèlisation <liste-de-stratégies>)  ||
                             (choix-système <liste-de-stratégies>)  ||
                             (choix-utilisateur <liste-de-strategies>) ||
                             (itération <étape>)                ||
                             (tâche-utilisateur <étape>)        ||
                             (exécution)                        ||
                             <étape>

<étape>                 :=   <symbole>                          ||
                             vide
```

Fig. 23 : Grammaire de description des stratégies de résolution

La figure précédente montre la grammaire de description de la stratégie de résolution à l'aide de ces opérateurs. Notez qu'une étape correspond en fait à une tâche spécifiée soit par son nom - lorsqu'on connaît explicitement le nom de la tâche à réaliser -, soit par l'action qu'elle réalise – lorsqu'on connaît l'action à réaliser, sans pour autant pouvoir nommer explicitement la tâche résolvante -.

Une étape (ou tâche) est caractérisée par un symbole (i.e. une chaîne de caractères) qui permet de faire le lien avec le contexte d'exécution de la tâche en question.

Les opérateurs de séquence, de parallèlisation, de choix et d'itération, sont ceux classiquement utilisés en algorithmique ; à ceux-là, nous avons ajouté des opérateurs permettant d'indiquer explicitement des phases de coopération système/utilisateur au cours de la résolution (choix_utilisateur, tâche_utilisateur).

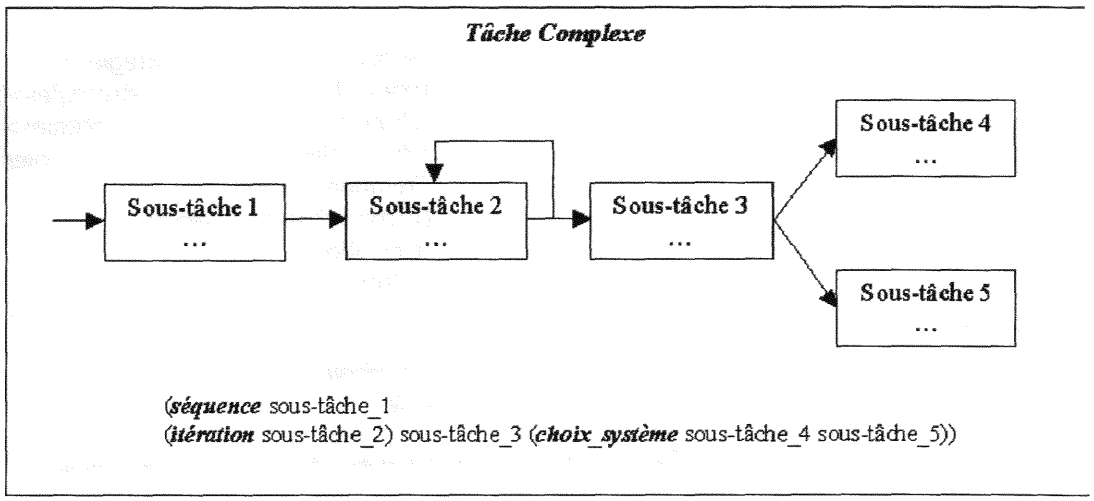


Fig. 24 : Exemple de description d'une stratégie de résolution

Les opérateurs de séquence, de parallèlisation et de choix (utilisateur ou système) s'appliquent sur un ensemble d'étapes de la résolution ; les opérateurs « itération » et « tâche utilisateur » ne concernent, quant à eux, qu'une seule étape.

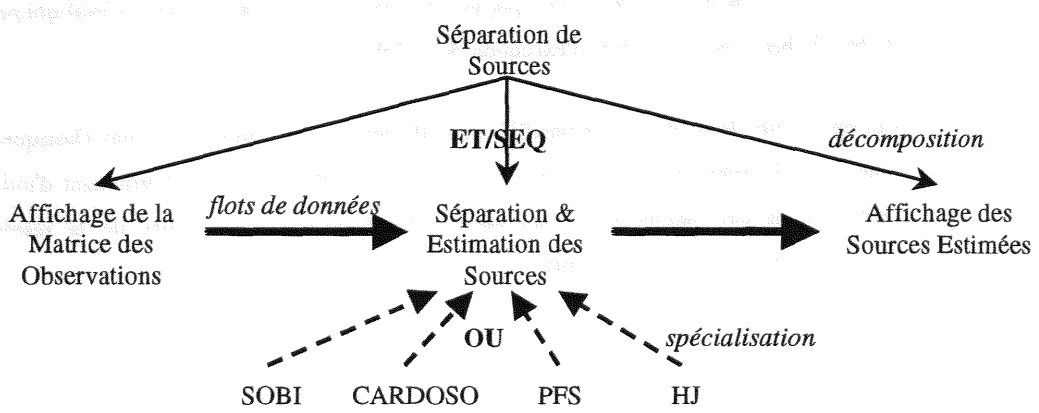


Fig. 25 : Stratégie de résolution associée à la séparation de sources dans notre maquette

B. Le contexte d'application d'une tâche

B.1. Avant-propos

Dans notre modèle, une tâche sera représentée par une classe d'objets. Comme précisé en introduction, l'ensemble des attributs de cette classe ou de toute sous-classe peut être séparé en deux blocs distincts. Le premier correspond au contexte d'application de la tâche ; il regroupe toutes les informations relatives à la classe de problèmes traitée, à savoir : ses entrées, ses sorties, la précondition de réalisation, la postcondition de réalisation, les tâches de visualisation associées et tous les attributs définis par l'utilisateur permettant de préciser des informations complémentaires.

S.C.D. - U.H.P. NANCY 1 SCIENCES BIBLIOTHÈQUE

Le second bloc correspond à la définition de la résolution de ce même problème : il s'agit là du contexte d'exécution de la tâche ; celui-ci sera décrit dans la section suivante.

B.2. Les entrées/sorties

Comme nous l'avons déjà vu, une tâche prend en entrée un certain nombre de données et produit, en contrepartie, un certain nombre de résultats. La description de ces entrées/sorties correspond à la description de l'interface externe d'une tâche. Les entrées d'une tâche sont considérées comme à priori indépendantes les unes des autres bien qu'il soit possible de définir des contraintes logiques entre elles (par ajout de préconditions).

Dans les entrées, nous incluons les paramètres de la tâche qui sont initialisés par le système avant la première exécution d'une tâche et éventuellement réajustés avant une nouvelle exécution (ce sont les paramètres des méthodes encapsulées par les tâches).

Les entrées et les sorties définies dans le contexte d'application d'une tâche correspondent, en fait, à l'interface de la tâche, c'est-à-dire à ce qui la caractérise du point de vue de l'extérieur.

B.3. Les préconditions

Il est possible de définir un ensemble de préconditions permettant de vérifier une ou plusieurs conditions pouvant porter sur les entrées et/ou sur les ressources disponibles (nécessaires à l'exécution de la tâche considérée). Celles-ci seront vérifiées une fois toutes les entrées fournies. Cela permet notamment de définir des dépendances logiques entre les entrées qui - comme nous l'avons déjà mentionné - sont considérées comme à priori indépendantes les unes des autres.

Les préconditions sont, en fait, encapsulées dans une tâche dont l'exécution a pour but leur évaluation.

B.4. Les postconditions

Il est possible de définir un ensemble de postconditions permettant de vérifier une ou plusieurs conditions pouvant porter sur tout ou partie des sorties. Celles-ci seront vérifiées une fois toutes les sorties calculées. Cela permet notamment de vérifier que les sorties obtenues sont bien conformes à ce qui est attendu.

Tout comme les préconditions, les postconditions renvoient à l'exécution d'une tâche.

B.5. La description des procédures (ou modules algorithmiques) associées

Les procédures sont utiles pour caractériser les tâches primitives qui se réduisent à leur exécution.

Il convient de distinguer deux types de procédures : les procédures internes qui décrivent un module de calcul intégré au système et les procédures externes qui décrivent un module de calcul externe - préexistant au système dont il est ici question -.

Les procédures internes

Les procédures internes sont définies à partir de leurs entrées, de leurs sorties et du code associé qui devra être exécuté.

Dans la définition d'une procédure, on trouvera les informations suivantes :

- le nom de la fonction ou de l'objet associé - suivant le langage support retenu -
- la liste des entrées et des sorties spécifiées comme pour tout autre tâche
- les attributs internes nécessaires à l'exécution de la procédure en question.

Par souci d'homogénéité, nous avons décidé que la fonction associée devra respecter le formalisme suivant : retourner ***1*** ou ***0*** selon que l'exécution s'est soldée par un ***succès*** ou par un ***échec***.

Les procédures externes

Une procédure externe ne diffère d'une procédure interne que par le fait que c'est une procédure qui n'a pas été écrite dans le langage de programmation de notre moteur de pilotage : c'est, par exemple, un des algorithmes préexistants que notre ***« Bibliothèque de Programmes Intelligente »*** devra gérer. Ainsi, une tâche peut à tout instant lancer l'exécution de programmes écrits en **Matlab**, en **C** ou en **FORTRAN**.

Cependant, ces programmes étant externes au système de pilotage et pouvant nécessiter de longs temps d'exécution, il faut pouvoir gérer leurs exécutions de façon asynchrone - c'est-à-dire de façon à ne pas bloquer le fonctionnement du moteur de pilotage -.

Remarque : dans le cadre de notre maquette, rien n'a été prévu pour la gestion asynchrone des exécutions.

Dans la définition d'une procédure externe, on trouve les informations suivantes :

- la chaîne de commande devant permettre le lancement du programme externe ;
- la liste des entrées/sorties spécifiées comme pour toute autre tâche ;

- tous les attributs internes nécessaires à l'exécution de la procédure en question.

C. La description de l'exécution

C.1. Avant-propos

Cette partie de la description d'une tâche renferme toutes les informations nécessaires à la réalisation de celle-ci : la description de la stratégie à suivre ainsi que le contexte de résolution lié à chacune de ses sous-tâches.

C.2. La stratégie de résolution

Rappels

La stratégie de résolution permet de spécifier comment une tâche est réalisée en terme de sous-tâches. C'est à partir de cette description à priori que le moteur de pilotage contrôle la réalisation de la tâche cible : sous-tâches exécutées en séquence, choix d'une sous-tâche parmi plusieurs en fonction du contexte, etc.

Cette stratégie est décrite grâce au langage permettant d'exprimer les différentes opérations présentées ci-avant (cf. définition d'une grammaire de description des stratégies de résolution) : le séquencement ou la parallèlisation d'opérations, le choix d'une sous-tâche spécifique, l'itération, etc.

Il faut noter que les opérateurs ***séquence***, ***parallèle***, ***choix_système*** et ***choix_utilisateur*** permettent de combiner d'autres opérateurs : ainsi les opérateurs de choix permettent de proposer plusieurs stratégies alternatives.

Les différents opérateurs et leur syntaxe

1. *L'opérateur séquence :* **seq**

Syntaxe : (***seq*** <stratégie> <stratégie> ...)

Cet opérateur permet de décrire qu'une suite de blocs stratégiques doit être exécutée en séquence. L'ordre des blocs étant l'ordre d'exécution, il ne peut pas être modifié dynamiquement par l'utilisateur ou par le système.

Cet opérateur effectue un ***ET*** logique entre ses opérandes : en effet, si l'exécution d'un des blocs stratégiques conduit à un échec, l'exécution de l'ensemble de la séquence échoue.

2. *L'opérateur parallèle :* **par**

Syntaxe : (***par*** <stratégie> <stratégie> ...)

Cet opérateur permet de décrire qu'un ensemble de blocs stratégiques doit être exécuté, mais que leur ordre d'exécution n'a pas d'importance. Ainsi, cela signifie que ces blocs stratégiques peuvent êtres réalisés en parallèle et qu'il n'existe aucune dépendance entre eux (en terme de flot de données notamment).

Il faut cependant signaler que l'échec de l'exécution d'un des blocs entraîne l'échec de l'exécution de l'ensemble des blocs, ce qui signifie que cet opérateur effectue lui-aussi un ***ET*** logique entre ses opérandes.

3. *Les opérateurs de choix :* **csy** *(choix_système)* / **cus** *(choix_utilisateur)*

Syntaxe : (***csy*** <stratégie> <stratégie> …) / (***cus*** <stratégie> <stratégie> …)

Ces opérateurs permettent d'indiquer qu'un choix doit être effectué (par le système ou par l'utilisateur suivant l'opérateur utilisé) entre plusieurs blocs stratégiques et/ou tâches.

Dans le cadre d'un choix utilisateur, c'est à l'utilisateur qu'incombe de choisir la stratégie ou la tâche qui lui semble être la mieux adaptée à ses besoins ; si son choix s'avère ne pas être le plus judicieux au vu des résultats, il devra pouvoir revenir en arrière, remettant ainsi en cause son choix infructueux.

Dans le cadre d'un choix automatique, le système pourra faire appel à divers modules d'aide à la décision : un système expert utilisant les connaissances recueillies auprès d'experts du domaine, un système de raisonnement à partir de cas utilisant les éventuelles analogies entre le problème en cours de traitement et des précédents problèmes déjà résolus, un système basé sur un modèle de décision permettant d'affecter à chaque option une utilité d'activation.

Cet opérateur effectue un ***OU*** logique entre ses opérandes : en effet, si tous les blocs stratégiques proposés lors du choix conduisent à un échec, alors la tâche en cours de résolution conduira elle aussi à un échec.

4. *L'opérateur itération :* **ite**

Syntaxe : (***ite*** <étape>) où une étape renvoie en fait à une tâche et à son contexte d'exécution.

Cet opérateur permet d'indiquer qu'une tâche (i.e. étape) doit être répétée jusqu'à ce qu'une condition soit vérifiée. L'échec de l'exécution de l'une des itérations conduira à l'échec de l'exécution du bloc stratégique en cours d'exploration ; en cela, cet opérateur effectue lui-aussi un ***ET*** logique entre les différentes itérations.

Cette itération peut être faite sur une liste de valeurs, sur un intervalle de valeurs ou jusqu'à ce qu'une condition explicite soit vérifiée. Une description plus détaillée des différents types d'itération et des informations qui leurs sont spécifiques est donnée dans la partie portant sur les contextes de résolution.

5. *L'opérateur itération_parallèle :* **itp**

Syntaxe : (***itp*** <étape>) où une étape renvoie là encore à une tâche et à son contexte d'exécution.

Cet opérateur permet d'indiquer qu'une tâche doit être répétée sur chacun des éléments d'une liste ou sur un intervalle, mais les différentes exécutions peuvent être réalisées en parallèle.

On considérera que si l'exécution de l'une des itérations conduit à un échec, alors le bloc stratégique en cours d'exécution échoue (***ET*** logique entre les itérations).

Cet opérateur a un intérêt tout particulier dans le cadre de l'utilisation de machines et d'algorithmes parallèles.

6. *L'opérateur tâche_utilisateur :* **ust**

Syntaxe : (***ust*** <étape>)

Cet opérateur permet d'indiquer que c'est l'utilisateur qui sera l'acteur de ce bloc stratégique. En fait, la tâche associée devra être liée à un contexte de type contexte d'interrogation (voir ci-après). Ce type de stratégie permet de décharger le système du calcul ou de l'estimation d'une valeur que l'utilisateur aura le soin de préciser.

7. *L'opérateur exécution :* **exec**

Syntaxe : (***exec***)

Cet opérateur permet d'indiquer au système que la tâche décrite par cette stratégie est une tâche élémentaire qui est donc directement associée à une méthode exécutable. Signalons à nouveau que cet opérateur ayant été ajouté postérieurement au développement de la maquette, il n'y apparaît pas.

C.1. Les contextes de résolution

Généralités

Chacune des sous-tâches décrites au sein de la stratégie de la tâche de résolution associée à la tâche englobante est spécifiée par un symbole qui renvoie à un des éléments de la liste des contextes d'exécution des sous-tâches associées à la tâche englobante ; cette liste permet de

décrire le contexte d'exécution de chaque sous-tâche dans le cadre de la réalisation de la stratégie permettant de résoudre la tâche englobante.

Le contexte de résolution d'une tâche est une entité permettant d'indiquer toutes les informations nécessaires à l'exécution d'une tâche. Ainsi, il permet de spécifier les flots de données, les conditions d'activation, les informations relatives à un type d'exécution (condition de fin d'itération, etc.) et toutes les informations permettant de moduler ou de spécifier le comportement du moteur de pilotage au moment de l'exécution, il inclut aussi la description de la sous-tâche à exécuter.

Le contexte d'exécution est spécifique du type de la tâche à exécuter. Ainsi, on peut déjà définir un certain nombre de types de contextes :

- ***contexte simple*** : il permet de décrire l'exécution d'une sous-tâche ;
- ***contexte d'interrogation*** : il décrit l'interrogation faite à l'utilisateur pour définir une valeur ;
- ***contexte d'itération conditionnelle*** : il décrit l'exécution de l'itération d'une tâche itérative jusqu'à ce qu'une certaine condition soit vérifiée ;
- ***contexte « essai-erreur »*** : il décrit l'exécution de l'itération d'une tâche itérative jusqu'à ce que l'utilisateur soit satisfait du résultat obtenu ;
- ***contexte d'itération sur une liste*** : il décrit l'exécution de l'itération d'une tâche itérative sur chaque valeur d'une liste ;
- ***contexte d'itération sur un intervalle*** : il décrit l'exécution de l'itération d'une tâche itérative sur chaque valeur d'un intervalle.

Quelque soit le type de contexte, il est nécessaire de spécifier un certain nombre de paramètres indispensables :

- le flot de données ;
- la tâche à exécuter.

De plus, il peut contenir d'autres informations relatives au contrôle de l'exécution telles que :

- le type de résolution : cela doit permettre de préciser si l'exécution de la tâche - et sa réussite - est optionnelle, soumise à une condition ou obligatoire ;

- la ou les conditions d'activation de la résolution : elle permet de spécifier une ou plusieurs conditions devant être vérifiées pour autoriser l'exécution de la tâche ;
- les paramètres de contrôle de l'exécution (validation, visualisation, etc.) qui permettent de moduler le comportement du moteur de pilotage.

La description du flot de données

Le flot de données permet de décrire les transferts d'information (i.e. de données) de la tâche englobante vers ses sous-tâches au moment de l'appel de ses dernières d'une part et réciproquement lors du renvoi des résultats obtenus par les sous-tâches exécutées vers la tâche englobante. De plus, il permet de décrire les échanges d'informations entre les différentes sous-tâches qui peuvent être appelées à communiquer entre-elles (c'est le cas par exemple pour des tâches devant s'exécuter en séquence).

Pour décrire les flots de données, nous avons été amenés à écrire une grammaire de description :

```
<liste-affectations>    :=    <affectation>; < liste-affectations > ||
                              vide
<affectation>           :=    <objet> = <objet>
<objet>                 :=    <identifiant>.<port>
<identifiant>           :=    symbole
<port>                  :=    symbole
```

Fig. 26 : Grammaire de description des flots de données

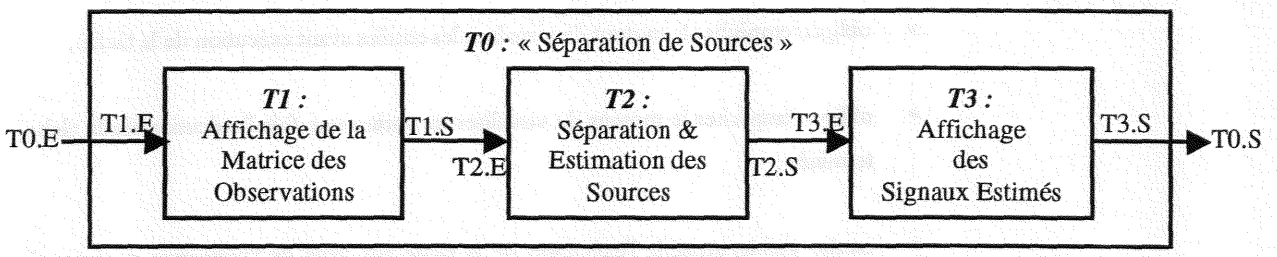


Fig. 27 : Schéma-bloc de la séparation de sources avec les flots de données

```
T1.E=T0.E ; T2.E=T1.S ; T3.E=T2.S ; T0.S=T3.S
```

Fig. 28 : Exemple de spécification des flots de données

Condition d'activation

La ou les conditions d'activation – qui renvoient elles-aussi à des tâches – permettent d'associer un ou plusieurs prédicats à une tâche de façon à autoriser ou à empêcher l'activation de la tâche. Suivant le type de résolution (conditionnelle, optionnelle ou obligatoire) le test associé à cet ensemble de prédicats aura des conséquences différentes.

- si la tâche est conditionnelle, la condition d'activation correspond à l'expression de la condition à vérifier pour que la tâche soit exécutée (mais dans tous les cas las résolution se poursuivra). Par exemple, dans le cadre de l'inversion de matrice, la tâche de diagonalisation ne sera exécutée que si la matrice n'est pas déjà diagonale ;
- si la tâche est optionnelle, la condition d'activation permet de filtrer les activations : si la condition n'est pas vérifiée, la tâche ne peut être exécutée, mais si la condition est vérifiée, l'exécution de la tâche doit être confirmée par l'utilisateur. L'activation ou l'échec de telles tâches n'entraîne aucunement l'échec de la résolution globale ;
- si la tâche est obligatoire, la condition indique la faisabilité ou l'échec de la résolution : si la condition est vérifiée, la tâche doit être exécutée, sinon la résolution échoue.

Contrôle de l'exécution

Il doit être possible de moduler le comportent du moteur lors de l'exécution d'une sous-tâche. Pour ce faire, on pourra définir un ensemble de paramètres optionnels permettant au modélisateur de modifier à priori le comportement du système.

On peut ainsi souhaiter mettre en place un certain nombre de comportements optionnels :

- obliger/empêcher le moteur d'afficher une boîte de dialogue demandant à l'utilisateur une validation des entrées et/ou des sorties ;
- obliger/empêcher le moteur de visualiser les entrées avant exécution de la tâche ;
- obliger/empêcher le moteur de visualiser les sorties une fois l'exécution de la tâche terminée ;
- rendre visible/invisible l'exécution de la tâche aux yeux de l'utilisateur de façon à autoriser/interdire les interactions entre l'utilisateur et le système pendant son exécution ; si une interaction est nécessaire alors que la tâche est invisible, l'exécution échouera.

A ces différents comportements, nous pourrons éventuellement en ajouter d'autres suivant nos besoins.

Le contexte d'une tâche itérative

Différents types de contexte itératif peuvent être définis pour répondre aux besoins des utilisateurs :

S.C.D. - U.H.P.
NANCY 1
SCIENCES
BIBLIOTHÈQUE

- itération conditionnelle : jusqu'à ce qu'une condition soit atteinte ;
- itération de type « essai-erreur » : jusqu'à ce que l'utilisateur soit satisfait du résultat qu'il a obtenu ;
- itération sur chaque élément d'une liste ;
- itération sur chaque élément d'un intervalle.

Chacun de ces différents contextes nécessite la description d'informations relatives à l'itération et au type de l'itération. Il est cependant possible d'isoler un certain nombre d'informations communes - i.e. valables pour ces différents types d'itérations - auxquelles nous allons tout d'abord nous intéresser.

Pour réaliser une itération, il peut être nécessaire d'effectuer un certain nombre de traitements :

- l'***initialisation*** qui permet de spécifier la méthode à exécuter afin d'initialiser certains paramètres de l'itération ou de calculer leurs valeurs ;
- le ***pré-traitement*** qui renvoie à une méthode permettant, avant chaque pas de l'itération, de faire un traitement spécifique (choix d'une nouvelle valeur pour un paramètre, etc.) ;
- le ***post-traitement*** qui renvoie à une méthode permettant, après chaque pas de l'itération, de faire un traitement spécifique (calcul d'une nouvelle valeur pour l'itération suivante, construction d'un résultat intermédiaire, etc.) ;
- la ***terminaison*** qui renvoie à une méthode permettant, après toutes les itérations, de faire un traitement spécifique (construction du résultat final à partir des résultats intermédiaires obtenus aux différentes itérations, etc.).

A ces attributs de description des méthodes associées, s'ajoute un attribut permettant de décrire le transfert des données à chaque nouvelle itération.

L'ordonnancement des actions au sein du système de pilotage sera le suivant :

1. lecture du flot de données global en entrée afin de récupérer les valeurs "externes" ;
2. activation de la méthode d'initialisation ;
3. exécution de la méthode de pré-traitement ;
4. lecture du flot de données en entrée spécifique à chaque itération ;

5. exécution de la tâche ;
6. lecture du flot de données en sortie spécifique à chaque itération ;
7. exécution de la méthode de post-traitement ;
8. reprise à l'étape 3 jusqu'à ce que la condition soit remplie ;
9. activation de la méthode de terminaison ;
10. lecture du flot de données global en sortie.

Le contexte d'une tâche itérative conditionnelle

La seule information à modéliser pour décrire une tâche conditionnelle en plus des informations générales communes à tous les types d'itération est la condition d'arrêt de l'itération. Celle-ci sera spécifiée à l'aide d'un attribut ***condition_d'arrêt***.

Le contexte de tâche « essai/erreur »

Le contexte "essai-erreur" doit permettre d'indiquer au système que c'est l'utilisateur qui précise la fin de l'itération : la condition est la satisfaction de l'utilisateur.

Le processus de résolution associé sera le suivant :

- exécution d'une tâche en précisant ses paramètres d'entrée ;
- validation par l'utilisateur du résultat acquis. Si l'utilisateur n'est pas satisfait du résultat, il peut adapter les paramètres avant de lancer une nouvelle itération ;
- dans le cas d'une non-validation, le système ré-exécute la tâche avec les paramètres modifiés par l'utilisateur ;
- dans le cas d'une validation, le système "exporte" le résultat validé.

Pour gérer ce processus, le contexte de résolution associé devra inclure deux nouveaux attributs permettant de décrire :

- la liste des valeurs-attributs que l'utilisateur devra valider ;
- la liste des paramètres que l'utilisateur est habilité à modifier avant une nouvelle itération.

Le contexte d'une tâche itérative sur une liste

Pour pouvoir gérer l'itération sur une liste, il suffit d'indiquer au système, en plus des informations générales liées aux itérations de tous types, sur quelles listes l'itération doit être

faite et pour quels attributs. On peut ainsi définir plusieurs listes associées à différents attributs ; la condition d'arrêt est alors qu'une des listes soit terminée.

Le contexte d'une tâche itérative sur un intervalle

Ce contexte permet simplement de faire une itération sur chaque valeur d'un intervalle borné avec un certain pas. Pour que le système soit apte à gérer ce type d'itération, il faut préciser un certain nombre d'attributs supplémentaires :

- la valeur de la borne inférieure de l'intervalle à parcourir ;
- la valeur de la borne supérieure de l'intervalle à parcourir ;
- la liste des attributs auxquels il faut transmettre les différentes valeurs pointées lors du parcours de l'intervalle : il faut indiquer au système vers quel attribut chacune des valeurs de l'intervalle sera transmise ;
- la valeur du pas entre chaque itération : on travaillera avec des pas entiers.

La condition de fin d'itération est le traitement de toutes les valeurs de l'intervalle.

Le contexte d'activation d'une procédure

La description du contexte d'activation d'une procédure est identique à celle d'une tâche. En fait, il n'y a aucune différence entre le fait d'exécuter un travail à l'aide d'une tâche ou à l'aide d'une procédure.

Le contexte d'interrogation

Ce type de contexte permet au modélisateur d'autoriser la demande d'une valeur à l'utilisateur pendant la résolution. Normalement, ces demandes concernent les paramètres d'entrée, mais il se peut que la valeur en question ne puisse être correctement déterminée par l'utilisateur qu'après avoir obtenu des résultats intermédiaires.

La tâche ou procédure associée à un tel contexte doit permettre le calcul d'une valeur possible des paramètres que l'utilisateur pourra ensuite modifier. On peut aussi permettre une visualisation des résultats intermédiaires de façon à aiguiller les décisions utilisateur.

Pour mettre en œuvre ce type de tâche, le système doit posséder les informations suivantes :

- le nom du ou des attributs que l'utilisateur sera appelé à définir ;

- la ou les valeurs fournies par l'utilisateur - avec éventuellement un renvoie vers une procédure de calcul ou de vérification de contraintes - ;

- la liste des attributs de la tâche englobante vers lesquels la ou les valeurs saisies doivent être renvoyées (i.e. description du flot de données associé).

Le contexte de visualisation

Ce type de contexte est destiné aux tâches de monitoring (i.e. de visualisation). Lorsque le système rencontre une telle tâche de monitoring, il se contente d'appeler la procédure de monitoring associée au contexte. La visualisation est achevée par une demande de validation par l'utilisateur, ce qui permet à ce dernier de constater une éventuelle erreur, une mauvaise appréciation ou n'importe quelle autre anomalie.

II. Notre Moteur de Pilotage

Dans cette partie, nous allons présenté notre formalisation du problèmes, ainsi que les deux architectures de pilotage que nous avons défini. En effet, dans un premier temps, nous nous sommes contentés d'une architecture centralisée (i.e. moteur de pilotage centralisé), mais très rapidement, celle-ci nous a semblé insuffisante pour pouvoir aborder des problèmes plus complexes nécessitant plus qu'un mécanisme élaboré d'exploration de stratégies plus ou moins pré-déterminées. Ainsi, nous avons opté pour une architecture distribuée qui nous permettre d'utiliser approche par modèle de tâches conjointement à une approche multi-agents, celle induite par le paradigme de la société de spécialistes.

A. Formalisation du problème

Soit $\mathcal{B} = \{0, 1\}$, l'ensemble des booléens ; $\mathcal{R}$, l'ensemble des requêtes ; $\mathcal{MT}$, l'ensemble des méta-tâches ; $\mathcal{T}$, l'ensemble des tâches ; $\mathcal{P}$, l'ensemble des problèmes ; $\mathcal{CR}$, l'ensemble des règles de choix ; $\mathcal{D}$, l'ensemble des données (i.e. des entités) et $\mathbb{N}_n = \{1, 2, 3, \ldots, n\}$, l'ensemble des n premiers entiers.

Définissons maintenant les différents types de connaissances que nous allons manipuler :

- un objet O est un couple ***(Nom, Valeur)***
- une entité E est un couple ***(Objet, Type)***
- les entrées/sorties (comme toutes les données manipulées) sont des entités ;
- une requête R est un triplet ***(Action But, Entrées, Sorties Attendues)*** ;
- une méta-tâche MT est un sextuplet ***(Action But, Entrées, Sorties, Tâches Associées, Règles De Choix, Règles d'Evaluation)*** ;
- une tâche T est un 10-uplet ***(Nom, Action But, Caractéristiques, Entrées, Sorties, Stratégie Associée, Règles De Choix, Règles d'Evaluation, Règles d'Adaptation)*** ;
- Un problème P est l'ensemble des requêtes que l'on souhaite satisfaire $\{R_i, i \in \mathbb{N}_n\}$.

Définissons maintenant les fonctions permettant la manipulation de ces entités :

ReplyToRequest : $\mathcal{R} \rightarrow \mathcal{MT}$

R → MT avec :

MT = {Ti / (Goal(Ti) = Goal(R)) ∧
Meets(Inputs (Ti), Inputs(R)) ∧
DataTypes(Inputs(Ti)) ⊂ DataTypes(Inputs(R)) ∧ DataTypes(Outputs(Ti)) = DataTypes(ExpectedOutput (R))}

Cette fonction permet de déterminer la méta-tâche associée à la requête en cours de traitement.

Meets : $\mathcal{D} \times \mathcal{D} \rightarrow \mathcal{B}$

(I_1, I_2) → B avec :

b = 1 si chaque élément de l'ensemble I_1 peut être mis en correspondance avec un élément de I_2
0 sinon

ChosenTask : $\mathcal{MT} \rightarrow \mathcal{T}$

MT → T avec :

T / T ∈ AssociatedTask(MT) ∧
BestFit (ChoiceRules(MT), Inputs(MT), T)

Cette fonction permet de déterminer la tâche la mieux adaptée au contexte courant.

BestFit: $\mathcal{CR} \times \mathcal{D} \times \mathcal{T} \rightarrow \mathcal{B}$

(CR, I, T) → B avec :

B = 1 si T est la tâche la plus appropriée au contexte d'exécution représenté par les règles de choix CR et les entrées I de la méta-tâche)
0 sinon

Solve : $\mathcal{P} \rightarrow \mathcal{T}^n$

P → Tn avec :

Tn = Solve(P)
= Solve({Ri, i∈ $\mathbf{N_n}$ })
= {MTi / ∀i∈ $\mathbf{N_n}$, MTi = ReplyToRequest(Ri)}
= {Ti / ∀i∈ $\mathbf{N_n}$, Ti = ChosenTask(MTi)}
= {Ti / ∀i∈ $\mathbf{N_m}$ ∃j∈ $\mathbf{N_n}$, SubTask(Ti, Tj)}

Cette fonction permet de résoudre un problème, c'est-à-dire de déterminer l'ensemble des tâches primitives à exécuter afin de résoudre le problème posé.

SubTask: $\mathcal{T} \times \mathcal{T} \rightarrow \mathcal{B}$

(T1, T2) → B avec :

B = 1 si T1 est une sous-tâche de T2 (càd une tâche apparaissant dans l'arbre de décomposition / spécialisation de T2)
0 sinon

B. Architecture centralisée

L'architecture centralisée qui est présentée ici est celle qui nous a servi de point de départ et qui nous a permis de valider notre modèle de tâches par le biais de notre maquette.

A.1. Architecture du moteur de pilotage

Description générale

Le moteur de pilotage doit fournir plusieurs fonctionnalités. Il doit, d'une part, permettre le choix des tâches à exécuter, d'autre part, l'exploration des stratégies associées, et, enfin, le contrôle de leur exécution.

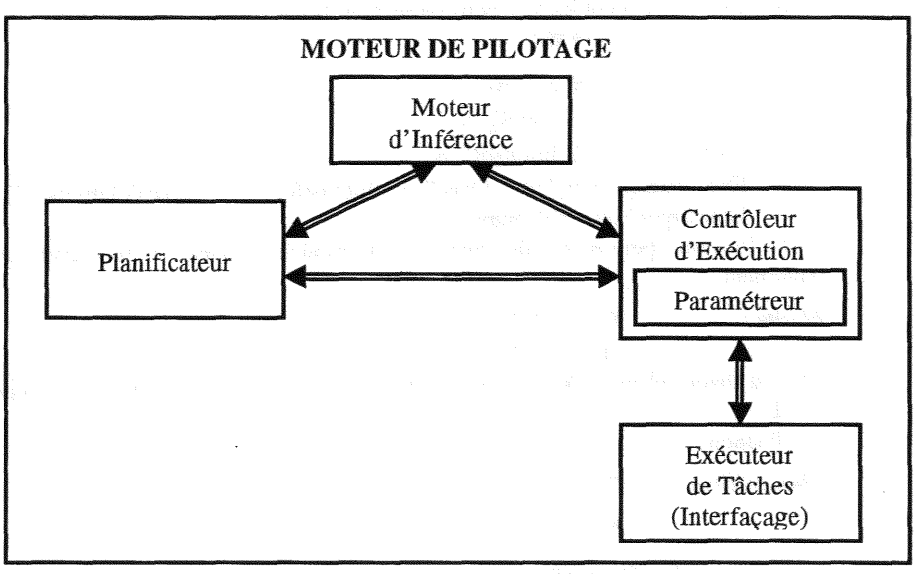


Fig. 29 : Architecture fonctionnelle de notre moteur de pilotage

Pour ce faire, le moteur est composé de différents modules de base : un composant pour chaque type d'activité : le planificateur pour sélectionner et ordonnancer les différentes tâches à utiliser pour réaliser l'action ciblée ; le contrôleur d'exécution pour initialiser/mettre à jour les paramètres d'exécution de ces tâches et pour contrôler les résultats de leurs exécutions ; l'exécuteur de tâches qui permet l'exécution des tâches primitives (celles directement associées à des modules exécutables) et qui doit donc résoudre les problèmes d'interfaçage ; le moteur d'inférence qui coopère avec le planificateur et le contrôleur d'exécution pour l'évaluation des règles de choix et d'évaluation des tâches et des règles d'initialisation et d'ajustement des paramètres.

Rôles des différents composants

Le planificateur

Le ***planificateur*** a pour rôle d'opérer la correspondance globale entre la Requête et la Méta-Tâche correspondante. Pour réaliser cette association, il se base sur l'action but et les entrées/sorties de ces dernières. Ensuite, il opère un pré-filtrage sur les Tâches associées à cette Méta-Tâches, en ne retenant que celles qui sont applicables au problème. Enfin, il ordonne les Tâches candidates et sélectionne celle - la plus adaptée au problème – qu'il enverra au contrôleur d'exécution. Ces phases de pré-filtrage, de classement et de choix se font à l'aide des règles de choix associée à la Méta-Tâche.

Remarque : en cas d'échec de la Tâche retenue – échec évalué à l'aide des règles d'évaluation qui permettent de juger les résultats fournis par la Tâche exécutée –, le planificateur passe à la Tâche candidate suivante.

Le contrôleur d'exécution

Le ***contrôleur d'exécution*** a pour rôle d'exécuter la Tâche que lui a transmise le planificateur – auquel, il renverra d'ailleurs, ensuite, le résultat produit par cette dernière -. Si cette Tâche est une tâche primitive, directement associée à une méthode exécutable, il l'envoie à l'***exécuteur de tâches*** pour exécution. Si, au contraire, cette Tâche est une tâche complexe, le contrôleur d'exécution aura pour rôle l'exploration de sa stratégie associée, faisant appel, au besoin – c'est-à-dire à chaque fois que dans celle-ci, une tâche sera désignée par son action but plutôt que par son nom – au planificateur pour élaborer les parties de stratégies pour lesquelles l'utilisateur n'avait pas de solution prédéterminée.

Une fois la Tâche qu'il a pour charge d'exécuter achevée, le contrôleur renvoie le résultat au planificateur pour évaluation.

Le paramétreur

Le ***paramètreur*** a pour rôle d'initialiser et, le cas échéant, d'adapter les paramètres de la tâche en cours d'exécution. Pour ce faire, il utilise les règles d'adaptation associées à cette dernière – en faisant appel au service du ***moteur d'inférence***.

L'exécuteur de tâches

Il a la charge de lancer et de surveiller l'exécution des méthodes - i.e. tâches primitives -, s'affranchissant au passage des problèmes d'interfaçage : il doit pouvoir lancer l'exécution d'une méthode interne ou externe, et ce avec les bons arguments. Une fois l'exécution achevée, il renvoie au contrôleur d'exécution le résultat de l'exécution (un booléen permettant de signaler le succès ou l'échec de l'exécution et, éventuellement, les données de sortie produites).

Remarque : à terme, l'exécuteur de tâche devra permettre d'interpréter directement des macro-commandes spécifiées par l'utilisateur dans la description des Tâches. Au préalable, il faudra, bien entendu, définir un langage de macro-commandes. On peut ainsi imaginer que l'utilisateur pourra directement associer à une Tâche, du code **JAVA** ou **C**.

Le moteur d'inférence

Le ***moteur d'inférence*** permet l'évaluation des règles de choix, d'évaluation et d'adaptation au fur et à mesure que cela s'avère nécessaire ; il est au service du planificateur et du contrôleur d'exécution. Ce moteur est générique, ce qui permet à l'expert de ne disposer que d'un seul formalisme (même syntaxe, mêmes structures de contrôle) pour toutes les bases de règles.

A.2. Mécanisme de contrôle

Introduction

Le moteur de pilotage travaille sur les objets Requêtes, Méta-Tâches, Tâches et Stratégies suivant le cycle de fonctionnement présenté sur la figure suivante :

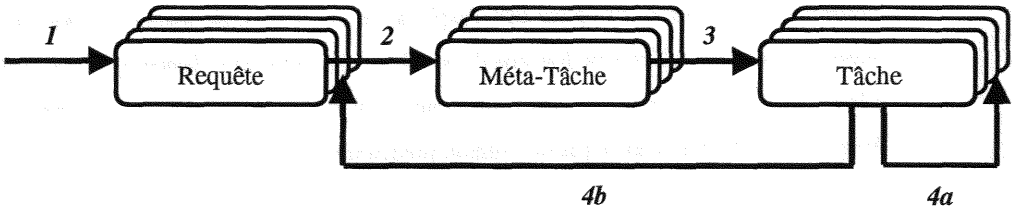


Fig. 30 : Cycle de fonctionnement du moteur de pilotage : planification hiérarchique et décomposition/spécialisation

A partir d'une Requête – celle initialement formulée par l'utilisateur ***(1)***, ou celle obtenue par décomposition/spécialisation d'une autre Tâche ***(4b)*** -, le système sélectionne ***(2)*** la Méta-Tâche associée. A partir de celle-ci, il est capable de sélectionner et d'ordonner les Tâches résolvantes ***(3)***, dont une seule sera retenue. La Tâche résolvante est alors soit directement exécutée – i.e. exécution de la méthode, du code ou de l'exécutable associé - s'il s'agit d'une Tâche Primitive, soit décomposée en sous-tâches ***(4a)*** ou sous-requêtes ***(4b)*** par le biais de sa Stratégie associée.

Principe de fonctionnement

Afin de démarrer le fonctionnement du système, l'utilisateur doit lui soumettre le problème auquel il veut s'attaquer par le biais d'une requête. Celle-ci lui permet d'exprimer le but qu'il cherche à atteindre par le biais de la description de l'action qu'il souhaite accomplir et des données d'entrées et de sorties qu'il désire que le système manipule et produise. A partir de cette requête, le système retrouve la méta-tâche associée; celle-ci renvoie alors à un ensemble de tâches concurrentes permettant de réaliser l'action but. La base de règles (règles de choix) associée à cette méta-tâche permet de sélectionner et d'ordonner les tâches adaptées au contexte suivant l'ordre imposé par le concepteur de la base de règles (critères de qualité, contraintes de ressources, etc.). Le système va alors exécuter la tâche la plus adaptée : exécution du module associé si la tâche est primitive, exploration de sa stratégie associée si elle est complexe. Si la tâche est complexe, elle peut ainsi renvoyer à d'autres tâches ou à d'autres requêtes qui seront récursivement exécutées/résolues.

Signalons qu'une tâche renverra à une requête, et non pas à une autre tâche, à chaque fois que l'utilisateur se verra dans l'incapacité de citer le nom de la tâche résolvante, tout en étant néanmoins capable d'identifier l'action à réaliser (ce qui permet l'expression d'un requête).

Enfin, lorsque plusieurs tâches résolvantes entrent en concurrence, le système, après les avoir ordonnées - à l'aide des règles de choix des méta-tâches ou des tâches selon qu'il s'agisse d'un choix implicite ou explicite c'est-à-dire intervenant ou non dans une stratégie -, les exécute les unes à la suite des autres jusqu'à ce que l'exécution de l'une d'entre elles se solde par une réussite ; cette réussite est évaluée à l'aide des règles d'évaluations associées aux méta-tâches et tâches.

Boucle de contrôle

Voici l'algorithme à implémenter dans la boucle de contrôle du Moteur de pilotage :

1. Recherche & ordonnancement de toutes les Requêtes & Tâches en attente ;

   TANT QU'il reste des Requêtes ou des Tâches non traitées :

   2. Choix d'une Requête ou d'une Tâche à traiter (cf. suivant leurs priorités) ;

      SI il s'agit d'une Requête **ALORS**

      a) Recherche de la Méta-Tâche associée ;

      b) Recherche & classement des Tâches résolvantes (pré-filtrage) ;

         TANT QUE la Requête en cours de traitement n'est pas satisfaite (i.e. tant que la Tâche résolvante choisie ne s'est pas soldée par un succès) :

         c) Choix de la « meilleure » Tâche résolvante en fonction du contexte ;

         d) Paramètrisation de la Tâche (initialisation/adaptation de ses paramètres) ;

         e) « Exécution » de la Tâche ;

            SI la Tâche est primitive **ALORS** exécution directe du Code/Module associé ;

            SINON décomposition en sous-tâches/sous-requêtes ;

      SINON

      a) Paramètrisation de la Tâche (initialisation/adaptation de ses paramètres) ;

      b) « Exécution » de la Tâche ;

SI la Tâche est primitive **ALORS** exécution directe du Code/Module associé ;

SINON décomposition en sous-tâches/sous-requêtes ;

3. Evaluation des résultats obtenus ;

C. Architecture distribuée (paradigme multi-agent de la « société de spécialistes »)

Après nous être intéressés à un certain nombre de travaux relatifs à la résolution de problèmes ([Lesser 77], [Lesser 88], [Lesser 93], [Lesser 94], [Marcos 98], [Moisan 97], [Parmentier 98], [Thonnat 93]), il nous a semblé opportun – et assez naturel -, de nous orienter vers une architecture multi-agents capable de manipuler notre modèle de tâches et de faire la synthèse entre les différents travaux en cours au sein de notre laboratoire (GTMAS : [Chevrier 93], CoMoMAS : [Glaser 96], JAVAMA : [Foisel 96] et [Foisel 97], ATOME : [Laasri 89], AITRAS : [Audrain 90a], [Audrain 90b], [Audrain 92a] et [Audrain 92b], G-SIGMA : [Gong 94], REAKT : [Mensch 93], [Mensch 95], [Mensch 96], PROGRESS : [Charpillet 97]). En effet, dans l'approche centralisée – présentée ci-avant –, on se contente d'explorer un arbre construit plus ou moins dynamiquement, mais toujours avec une approche descendante (i.e. guidée par les buts): on part du but initial qu'on décompose ensuite en sous-buts à atteindre, et ce récursivement. Avec notre approche multi-agents, nous souhaitons réintroduire une approche ascendante (i.e. guidée par les données) en permettant aux agents, en fonction des données qu'ils observent, de se proposer pour résoudre telle ou telle tâche intervenant dans le processus global de résolution.

C.1. Pourquoi utiliser une approche multi-agents

Le principe général des systèmes multi-agents est de partager et de distribuer entre plusieurs agents (intelligents ou non) l'ensemble des connaissances et la capacité de raisonnement que possède un système intelligent. Chacun de ces agents est spécialisé dans un sous-domaine du domaine de départ. Il est à même, de par l'expertise qu'il renferme, de résoudre tout ou partie du problème initial si les données et les ressources dont il dispose le lui permettent. La collaboration avec les autres agents du système lui permet d'améliorer sa propre participation à la résolution du problème global, de compléter les informations dont pourraient avoir besoin les autres agents et d'agir au moment opportun.

C.2. Présentation du paradigme retenu

Comme dans notre société où une compétence est associée à un ensemble de spécialiste du domaine, nous avons choisi d'associer une compétence à chaque agent du système ; ainsi, lorsqu'un problème se pose, les agents pouvant le résoudre se proposent et c'est alors à l'agent

spécialiste choisi de prendre en charge la résolution du problème qui lui a été confié (décentralisation du contrôle).

L'idée directrice est donc que chaque agent a son propre mécanisme de contrôle (i.e. son moteur de pilotage) afin qu'il puisse prendre en charge la « partie » de la résolution du problème global pour laquelle il s'est engagé à fournir une solution.

C.3. Extension « multi-agents » de notre modèle de tâches et de notre moteur de pilotage

Comme dans le cadre d'un contrôle centralisé (cf. architecture présentée ci-avant), nous allons avoir besoin des mêmes structures de données : les Requêtes, les Méta-Tâches et les Tâches.

Par contre, les structures de Méta-Tâches pourront désormais être encapsulées par des Agents qui auront, dès lors, la responsabilité du bon déroulement de la partie de la résolution associée.

Chaque agent aura donc son propre moteur de pilotage, identique à celui présenté dans le chapitre précédent (même cycle de fonctionnement, même principe de fonctionnement) afin d'assurer le contrôle de la résolution du sous-problème qui est à sa charge.

C.4. Processus de résolution

La Requête utilisateur initiale étant postée, l'Agent Résolveur va choisir parmi les agents candidats, celui le plus apte à résoudre le problème ; pour ce faire, il dispose de règles de choix lui permettant d'évaluer (et donc d'ordonner) les différents candidats. L'agent retenu pourra alors commencer la résolution en confiant éventuellement certaines sous-parties de la résolution à d'autres agents de sorte à sous-traiter certaines parties du problème.

L'agent qui a sous-traité tout ou partie de la résolution est chargé, après récupération des résultats de cette sous-traitance, d'évaluer ceux-ci et, en cas de résultats infructueux, il pourra confier cette même sous-tâche à un autre des ses candidats sous-traitants.

Ce mécanisme de sous-traitance peut, bien sûr, être récursivement utilisé par les agents sous-traitants eux-mêmes afin de leur permettre de se décharger d'une partie du problème dont la résolution leur incombe.

Dans un premier temps, on envisagera une collaboration du type : appel d'offre/sous-traitance, mais à terme, la stratégie de collaboration (i.e. le schéma interactionnel entre les agents) pourra être paramétrable (cf. travaux de Rémy FOISEL : [Foisel 96], [Foisel 97], [Foisel 98]) ce qui pourra permettre au système d'adapter dynamiquement sa stratégie de résolution au fil de la résolution en cours.

La figure suivante illustre le mécanisme de résolution présenté ci-avant :

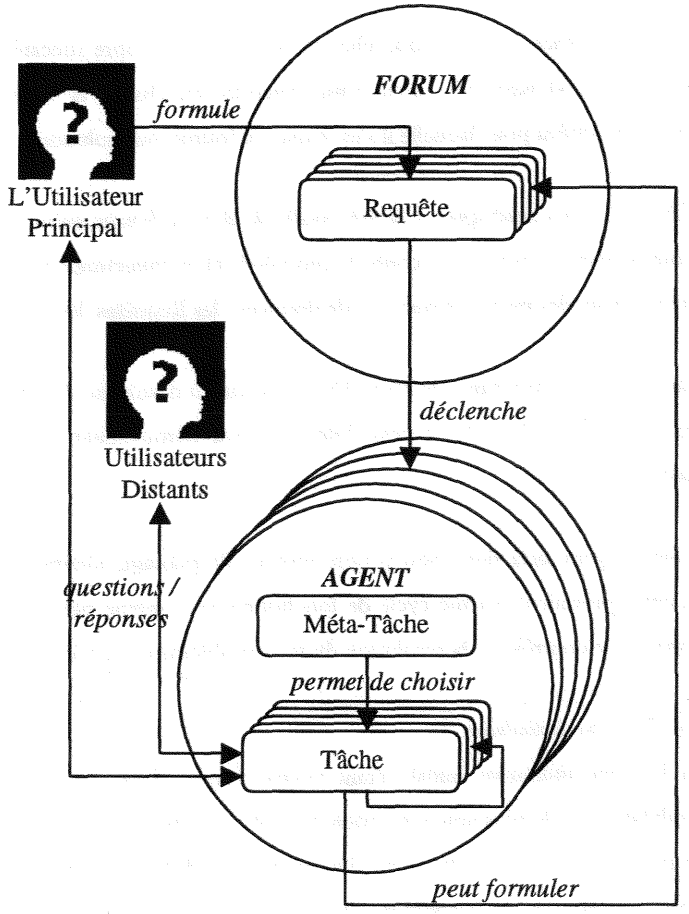


Fig. 31 : Cycle de fonctionnement dans le cadre du paradigme multi-agents

C.5. Architecture du moteur de pilotage

Comme nous l'avons déjà dit, dans cette approche « multi-agents », chaque agent (i.e. spécialiste) du système (i.e. de la société) aura son propre moteur de pilotage construit à l'image de celui dont l'architecture a été présentée dans le cadre de l'approche centralisée. Pour la description de ces moteurs de pilotage et de leurs composants, le lecteur est donc invité à se référer au paragraphe II.B.1.

D. Conclusion

L'architecture centralisée de notre moteur de pilotage a été maquettée, utilisée et testée, dans le cadre de l'application construite autour de la séparation de source. Les résultats obtenus sur des signaux tests ont étés ceux attendus, mais étant donné la « simplicité relative » du problème en terme de choix de méthodes – les critères de choix d'une méthode parmi

S.C.D. - U.H.P.
NANCY 1
SCIENCES
BIBLIOTHÈQUE

l'ensemble des méthodes disponibles étant exclusifs –, il conviendrait d'utiliser notre moteur au sein d'une application plus complexe afin d'en tester la validité.

L'architecture distribuée, quant à elle, n'a pas encore été utilisée, mais son intérêt nous semble indéniable dans le cadre général de notre problématique et plus spécifiquement dans le contexte de notre laboratoire où des travaux autour des mécanismes d'interaction (établissement et recombinaison dynamique des interactions entre agents) sont en cours et où les acquis en terme de systèmes multi-agents sont indéniables.

Chapitre 3

REALISATION

Application à la Séparation de Sources

Afin de tester notre modèle de tâches et notre approche basée sur le pilotage de programmes, nous avons entrepris de mettre en chantier une maquette autour d'un problème relativement simple, celui de la séparation de sources. C'est donc cette application que nous vous présenterons dans ce chapitre.

I. Qu'est-ce-que la séparation de sources

A. Présentation du problème

A.1. Généralités

Dans de nombreux domaines tels que l'accoustique, la sismique, la résistance des matériaux, le biomédical, on utilise les observations extérieures d'un phénomène – i.e. les signaux reçus sur un réseau de capteurs - pour caractériser leur cause - un ou plusieurs émetteurs qui sont à l'origine des signaux observés -. En effet, le signal observé à la sortie de chaque capteur est lié aux différentes sources qui sont à son origine : ainsi, ce signal observé reflète un mélange des sources. La séparation de sources vise à retrouver, le plus précisément possible, les sources à partir des signaux observés. Cette séparation s'effectue, en outre, sans connaissance à priori, ni sur les processus observés, ni sur les signaux sources, mais utilise des propriétés statistiques des sources, telle que leur indépendance – même si, en pratique, il est difficile de dire si les sources sont réellement indépendantes étant donné qu'elles « baignent » dans le même milieu : influence climatique, etc. -.

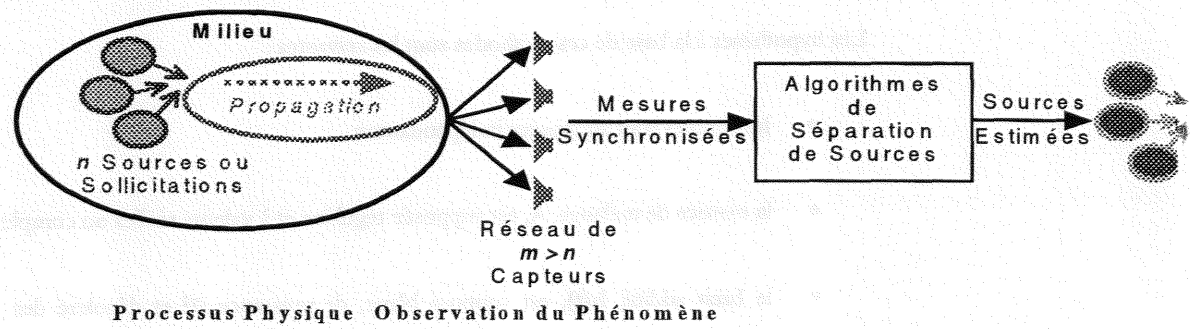


Fig. 32 : Qu'est-ce que la séparation de sources ?

A.2. Hypothèse simplificatrice

Afin de modéliser le problème de la séparation de sources, nous adopterons un modèle simplifié du milieu : nous le considérerons parfait, c'est-à-dire de gain constant dans la bande

de fréquences étudiée. En pratique, cette hypothèse permet d'aborder un grand nombre de cas « concrets ».

B. Cas du mélange instantané

B.1. Modélisation

On cherche donc à déterminer les signaux-sources en l'absence de toute autre information à priori sur la forme du front d'onde (modélisé par la forme de la matrice **A**), sur la disposition des ***m*** capteurs ou sur la distribution des ***n*** sources. Les méthodes de séparation de sources présentées dans ce document reposent donc uniquement sur l'hypothèse ***d'indépendance statistique des différentes sources.***

Le signal reçu à l'instant ***t*** peut alors s'écrire :

$$y(t) = \begin{pmatrix} y_1(t) \\ y_2(t) \\ \dots \\ y_m(t) \end{pmatrix} = A.s(t) + b(t) = \begin{pmatrix} a_{1,1}\ a_{1,2}\ a_{1,3}\ \dots\ a_{1,n} \\ a_{2,1}\ a_{2,2}\ a_{2,3}\ \dots\ a_{2,n} \\ \dots\ \dots\ \dots\ \dots\ \dots \\ a_{m1}\ a_{m,2}\ a_{m,3}\ \dots\ a_{m,n} \end{pmatrix} \begin{pmatrix} s_1(t) \\ s_2(t) \\ \dots \\ s_n(t) \end{pmatrix} + \begin{pmatrix} b_1(t) \\ b_2(t) \\ \dots \\ b_m(t) \end{pmatrix}$$

avec :

- **y(t)**, le vecteur des observations, de dimension ***(m, 1)*** ;
- **A**, la matrice de mélange, de dimension ***(m, n)*** ;
- **s(t)**, le vecteur des sources, de dimension ***(n, 1)*** ;
- **b(t)**, le vecteur de bruit additif, de dimension ***(m, 1)***.

Les hypothèses à la base de ces méthodes sont les suivantes :

- les sources sont supposées indépendantes;
- la matrice de mélange, **A**, est supposée régulière et à valeurs réelles ou complexes ;
- le bruit additif, **b(t)**, est supposé blanc, de puissance σ^2 et décoléré des signaux sources.

Les méthodes de séparation de sources opèrent donc une transformation du type :

s'(t) = B . y(t) où :

- **s'(t)** représente l'estimée du vecteur source **s(t)** ;

- **B** est la matrice séparante, de dimension ***(n, m)*** ;

Remarques :

- la séparation est donc achevée lorsque chaque signal-source est estimé à un facteur scalaire près ;

- les signaux-sources sont restitués indifféremment sur les différentes voies du séparateur (i.e. l'estimée des signaux-sources est le vecteur sources à une permutation près) ;

- dans le contexte « aveugle » dans lequel nous travaillons (aucune information à priori sur le milieu), une identification complète de **A** est exclue ; il en résulte donc une indétermination sur l'amplitude des sources.

B.2. Les algorithmes « blocs »

Introduction

Comme nous l'avons déjà signalé, le but des différentes méthodes de séparation de sources est de calculer une matrice séparante **B** afin de pouvoir remonter jusqu'aux signaux-sources. Les algorithmes ***« blocs »*** sont appelés ainsi parce qu'ils traitent la globalité du signal observé pour estimer **B**.

Les algorithmes « blocs » mis à notre disposition par la Division Recherche & Développement (anciennement Direction des Etudes et Recherches) d'EDF sont :

- méthode CARDOSO d'après l'algorithme proposé par A. Souloumiac et J.F. Cardoso ;

- méthode SOBI (Second Order Blind Identification) d'après l'algorithme proposé par A. Belouchrani.

Principe

Ces deux méthodes se composent toutes les deux de deux étapes :

- une étape de blanchiment du signal d'observation qui consiste en une normalisation des signatures et en une projection sur l'espace signal (une des conséquences de cela est que la matrice de mélange devient unitaire) ;

- une étape de séparation des signaux basée sur la détermination de cette matrice unitaire par une rotation qui rend les signaux blanchis indépendants

Remarque : SOBI met en jeu des statistiques d'ordre deux, alors que CARDOSO est basé sur des statistiques d'ordre quatre.

Algorithme SOBI

Hypothèses de la méthode

- les sources doivent posséder des cohérences temporelles différentes ;

- les observations doivent être à valeurs réelles ou complexes.

Contrainte supplémentaire

- en supposant que le milieu est de gain unitaire, on peut estimer la puissance des sources en normant les colonnes de la matrice de mélange **A**.

Choix pratiques (réglage des paramètres)

- le nombre de sources est le paramètre essentiel et doit être correctement dimensionné ; par défaut, ce nombre pourra soit être pris égal au nombre de capteurs m - car $n \leq m$ -, soit être estimé à l'aide de l'une ou l'autre des méthodes dont nous disposons (algorithme AIC_MDL ou, à défaut, algorithme LATOMBE) ;

- le nombre de matrices à diagonaliser est le second paramètre ; il correspond au nombre de retards et peut donc être choisi en mesurant la durée de corrélation des observations (avec notre algorithme COHERENCE) ; il est toutefois important de signaler qu'il vaut mieux surévaluer ce paramètre.

Algorithme CARDOSO

Hypothèses de la méthode

- les sources doivent être non gaussiennes (i.e. circulaires dans le cas complexe) ;

- les observations doivent être à valeurs réelles ou complexes.

Contrainte supplémentaire

- en supposant que le milieu est de gain unitaire, on peut estimer la puissance des sources en normant les colonnes de la matrice de mélange **A**.

- le nombre de sources est le seul paramètre ; là encore c'est un paramètre essentiel et il doit donc être correctement dimensionné ; par défaut, ce nombre pourra soit être pris égal au nombre de capteurs m - car $n \leq m$ -, soit être estimé à l'aide de l'une ou l'autre des méthodes dont nous disposons (algorithme AIC_MDL ou, à défaut, algorithme LATOMBE).

B.3. Les méthodes adaptatives

Introduction

Les algorithmes adaptatifs estiment eux aussi les signaux-sources à partir des signaux observés, mais contrairement aux algorithmes « blocs », il le font pour chaque pas de temps.

Principe

La séparation de sources adaptatives est modélisée par le schéma suivant :

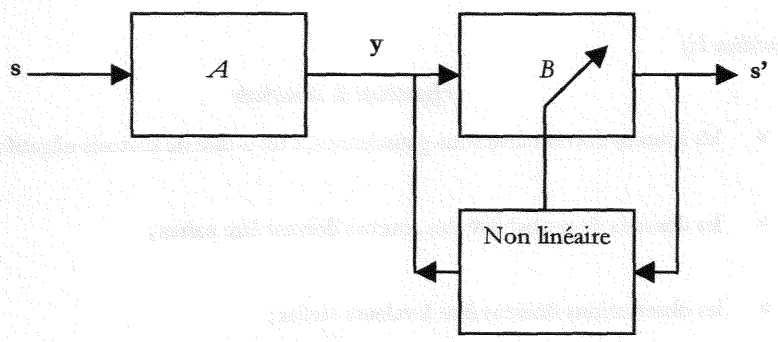


Fig. 33 : Principe d'un algorithme adaptatif

Dans cette approche, la matrice **B** est adaptée au fil du temps afin d'obtenir des signaux statistiquement indépendants à partir des signaux observés par le biais des capteurs.

Il faut signaler que dans ce modèle, on ne fait pas intervenir de bruit additif, car on ne peut pas prendre en compte de façon simple ce type de perturbation.

L'atout de ces méthodes adaptatives de séparation de sources réside dans leur simplicité d'implémentation.

Les algorithmes adaptatifs sont regroupés par les spécialistes en deux grandes catégories suivant que leurs performances sont uniformes ou non ; le terme de performances « uniformes » indiquant qu'en l'absence de bruit, l'erreur sur l'estimation des sources est identique – i.e. uniforme – quelle que soit la matrice de mélange **A**. A contrario, l'expression « non uniforme » indique que cette erreur est fortement dépendante de la matrice de mélange.

Les algorithmes adaptatifs mis à notre disposition par la Division Recherche & Développement d'EDF sont :

- méthode HJ d'après l'algorithme proposé par C. Jutten et J. Hérault ;
- méthode PFS d'après l'algorithme proposé par B. Lahed.

B. Lahed qui a comparé dans sa thèse différentes méthodes adaptatives de séparation de sources conclut que :

- HJ a des performances non uniformes et que même si les performances sont de bonne qualité pour le mélange étudié, la convergence de cet algorithme est relativement lente ;
- PFS a des performances uniformes.

Algorithme HJ

Hypothèses de la méthode

- les sources doivent être sous-gaussiennes, c'est-à-dire de kurtosis négatifs ;
- les densités de probabilité des sources doivent être paires ;
- les observations doivent être à valeurs réelles ;
- le nombre de sources doit être égal au nombre de capteurs ;
- le bruit est supposé négligeable, ce qui permet d'écrire : $\mathbf{y(t) = A \, . \, s(t)}$.

Si l'une des deux premières conditions n'est pas respectée, l'algorithme possèdera des points de convergence parasites.

Contrainte supplémentaire

- On désire restituer la puissance des sources ; pour ce faire HJ impose que la matrice séparante **B** soit de la forme suivante :

$$\mathbf{B = (Id + C)^{-1}}$$

avec : $\mathbf{C = [c_{i,j}]}$ et $\forall \mathbf{i}, \mathbf{c_{i,i} = 0}$ et : **Id** est la matrice identité.

Choix pratiques (réglage des paramètres)

- une initialisation à zéro des coefficients $c_{i,j}$ est satisfaisante pour assurer la convergence de l'algorithme (cela a été mathématiquement prouvé) ;

- le pas d'adaptation doit être inférieur à un ; il est, en général, de l'ordre du dixième. Pour le régler, on sait qu'il faut qu'il soit assez grand pour assurer la convergence, mais pas trop grand pour éviter une trop grande variance dans l'estimation de la matrice de mélange au fil du temps.

Remarques :

- les performances de l'algorithme sont insuffisantes lorsque les puissances des sources sont trop différentes (15 à 20 dB de différence) ;

- la convergence est lente (quelques centaines à quelques milliers de points sont nécessaires) ;

- l'algorithme est extrêmement sensible au bruit de mesure (conséquence directe des hypothèses).

Algorithme PFS

Hypothèses de la méthode

- les sources doivent être non gaussiennes et leurs kurtosis doivent être de même signe (la version dont nous disposons impose des kurtosis tous négatifs) ;

- les observations doivent être à valeurs réelles ou complexes ;

- le bruit est supposé négligeable, ce qui permet d'écrire : $\mathbf{y(t) = A \,.\, s(t)}$.

Remarque : cet algorithme est la version adaptative des algorithmes « blocs » présentés ci-avant ; il combine donc des étapes de blanchiment et de rotation.

Contrainte supplémentaire

- *Aucune* : les sources sont de puissances unitaires à la sortie du séparateur ; on obtient donc les sources à un facteur près.

Choix pratiques (réglage des paramètres)

- cet algorithme doit être initialisé par une matrice $\mathbf{B_0}$ assez proche de la matrice séparante que l'on cherche à estimer ; il peut donc être souhaitable d'utiliser un

algorithme « bloc » parmi ceux présentés ci-avant afin de l'initialiser. On peut faire remarquer à cette occasion, qu'à défaut d'en avoir une meilleure estimation, on peut prendre la matrice identité comme matrice $\mathbf{B_0}$, ce qui permet, en général, d'assurer la convergence ;

- le pas d'adaptation doit être inférieur à un ; il est souvent de l'ordre du centième. Pour le régler, on sait qu'il faut qu'il soit assez grand pour assurer la convergence, mais pas trop grand pour éviter une trop grande variance dans l'estimation de la matrice de mélange **B** au fil du temps.

Remarques : on peut donc conclure que les algorithmes « blocs », bien que plus complexes et coûteux que les algorithmes adaptatifs, sont plus largement utilisables du fait de leurs hypothèses moins contraignantes.

S.C.D. - U.H.P. NANCY 1 SCIENCES BIBLIOTHÈQUE

C. Choix d'une méthode de séparation de sources

C.1. Dans le cadre de signaux supposés stationnaires

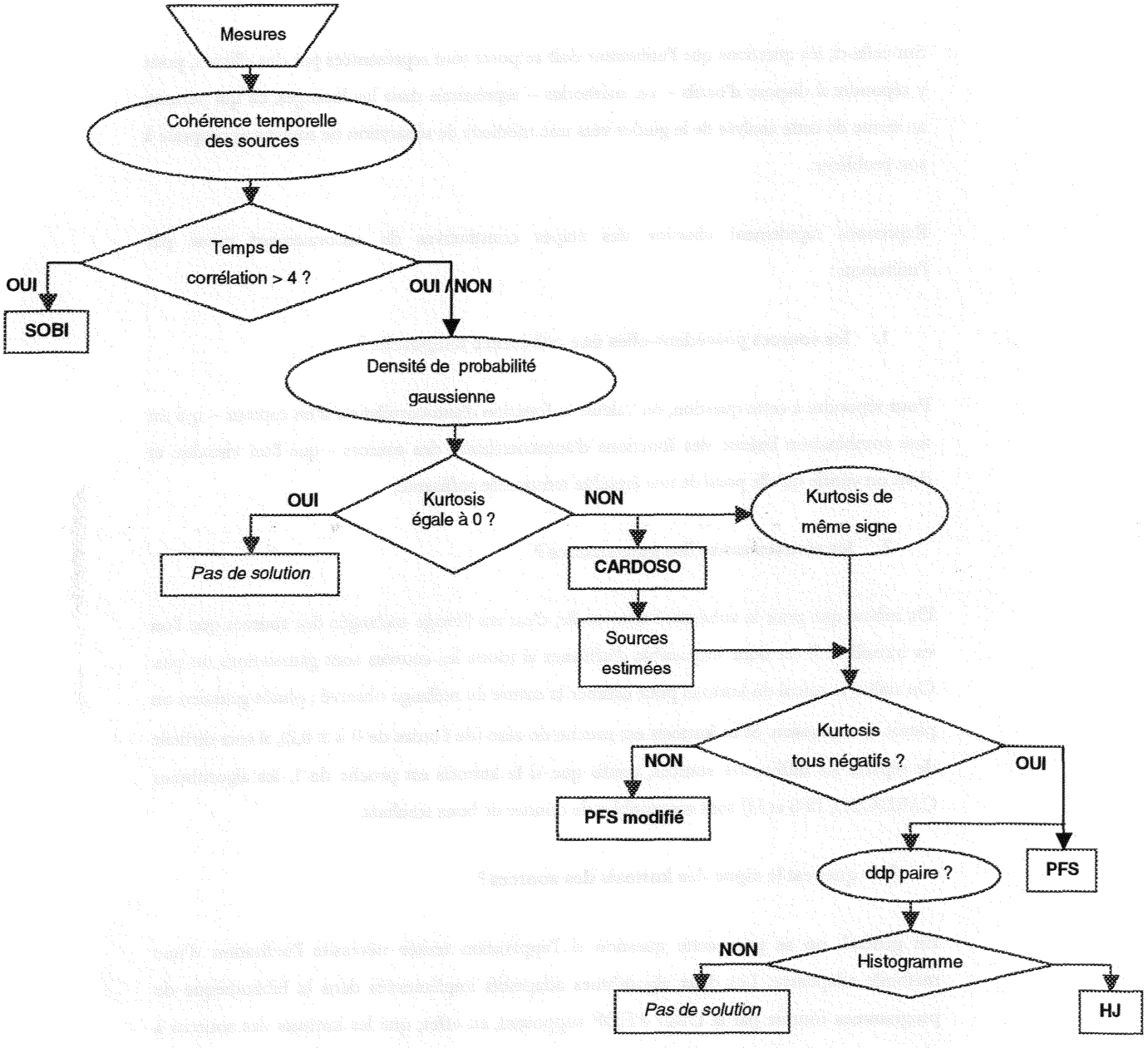


Fig. 34 : La démarche initiale de choix d'un algorithme de séparation de sources

La figure de la page précédente présente l'arbre de décision proposé par Guy d'Urso dans son rapport intitulé « Qu'est-ce que la séparation de sources ».

Cette figure illustre la démarche qu'il faut adopter lorsqu'on cherche à utiliser les algorithmes de séparation de sources de la bibliothèque que Guy d'Urso a développée et qui est composée des algorithmes SOBI, CARDOSO, HJ et PFS.

Sur celle-ci, les questions que l'utilisateur doit se poser sont représentées par des ellipses, pour y répondre il dispose d'outils – i.e. méthodes – représentés dans les losanges, ce qui permet, au terme de cette analyse de le guider vers une méthode de séparation de sources appropriée à son problème.

Reprenons rapidement chacune des étapes constitutives du raisonnement mené par l'utilisateur :

1. **les sources possèdent-elles une cohérence temporelle ?**

Pour répondre à cette question, on calcule la fonction d'autocorrélation d'un capteur – qui est une combinaison linéaire des fonctions d'autocorrélation des sources - que l'on visualise et dont on vérifie qu'elle possède une étendue temporelle suffisante.

2. **les sources sont-elles gaussiennes ?**

De même que pour la cohérence temporelle, c'est sur l'image mélangée des sources que l'on va travailler : il est donc impossible d'affirmer si toutes les sources sont gaussiennes ou pas. On utilise le calcul du kurtosis pour estimer la nature du mélange observé : plutôt gaussien ou plutôt non-gaussien. Si ce kurtosis est proche de zéro (de l'ordre de 0 à $\pm$ 0,2), il sera difficile de séparer les différentes sources, tandis que si le kurtosis est proche de 1, les algorithmes CARDOSO, PFS et HJ sont susceptibles de donner de bons résultats.

3. **quel est le signe des kurtosis des sources ?**

En général, on se pose cette question si l'application traitée nécessite l'utilisation d'une méthode adaptative. Les deux algorithmes adaptatifs implémentés dans la bibliothèque de programmes fournie par la DRD d'EDF supposent, en effet, que les kurtosis des sources à séparer sont tous de signe négatif. Cette condition n'est toutefois pas totalement restrictive, car il est possible de modifier l'algorithme PFS pour qu'il converge correctement si les sources à séparer sont toutes de kurtosis positifs.

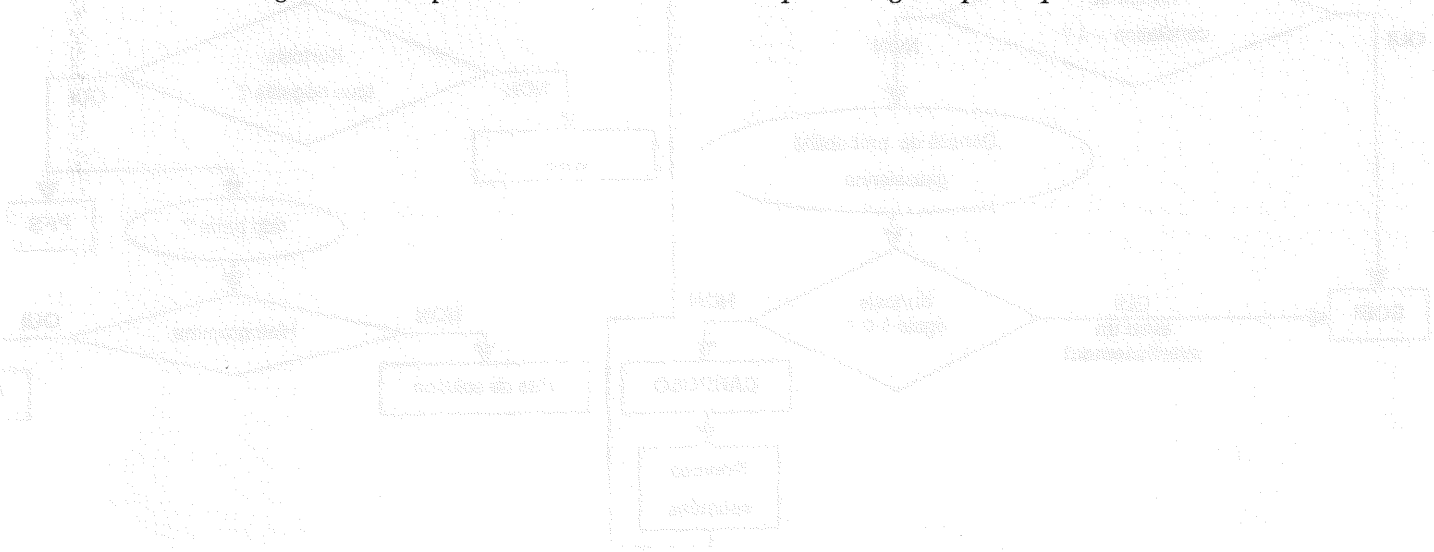
4. les densités de probabilités des sources sont-elles paires ?

Il est aisé de vérifier cette propriété en traçant l'histogramme des sources préalablement séparées en utilisant l'algorithme CARDOSO. Il importe également de le re-vérifier après l'utilisation de l'algorithme HJ, car l'omission de cette condition peut faire converger l'algorithme vers un point d'équilibre donnant des solutions fausses.

Remarque : Ce graphe a initialement été proposé dans le cadre de la séparation de sources stationnaires (sources n'évoluant pas dans le temps). Nous verrons plus tard l'extension qu'il a récemment proposée afin d'élargir le champ d'action de ces méthodes au cas des signaux non stationnaires.

C.2. Dans le cadre de signaux non supposés stationnaires

Guy d'Urso a récemment été amené à étendre son étude afin de pouvoir utiliser les algorithmes de sa bibliothèque (tout du moins, ceux qui s'y prêtent bien) sur des signaux non stationnaires. Ainsi, la figure suivante présente le nouvel arbre de décision proposé par Guy d'Urso; celui-ci présente la démarche qu'il faut adopter lorsqu'on cherche à utiliser les algorithmes de séparation de sources de sa bibliothèque a des signaux quelconques.



Remarque : sur la figure suivante, les mêmes conventions de représentation que celles précédemment utilisées ont été reprises.

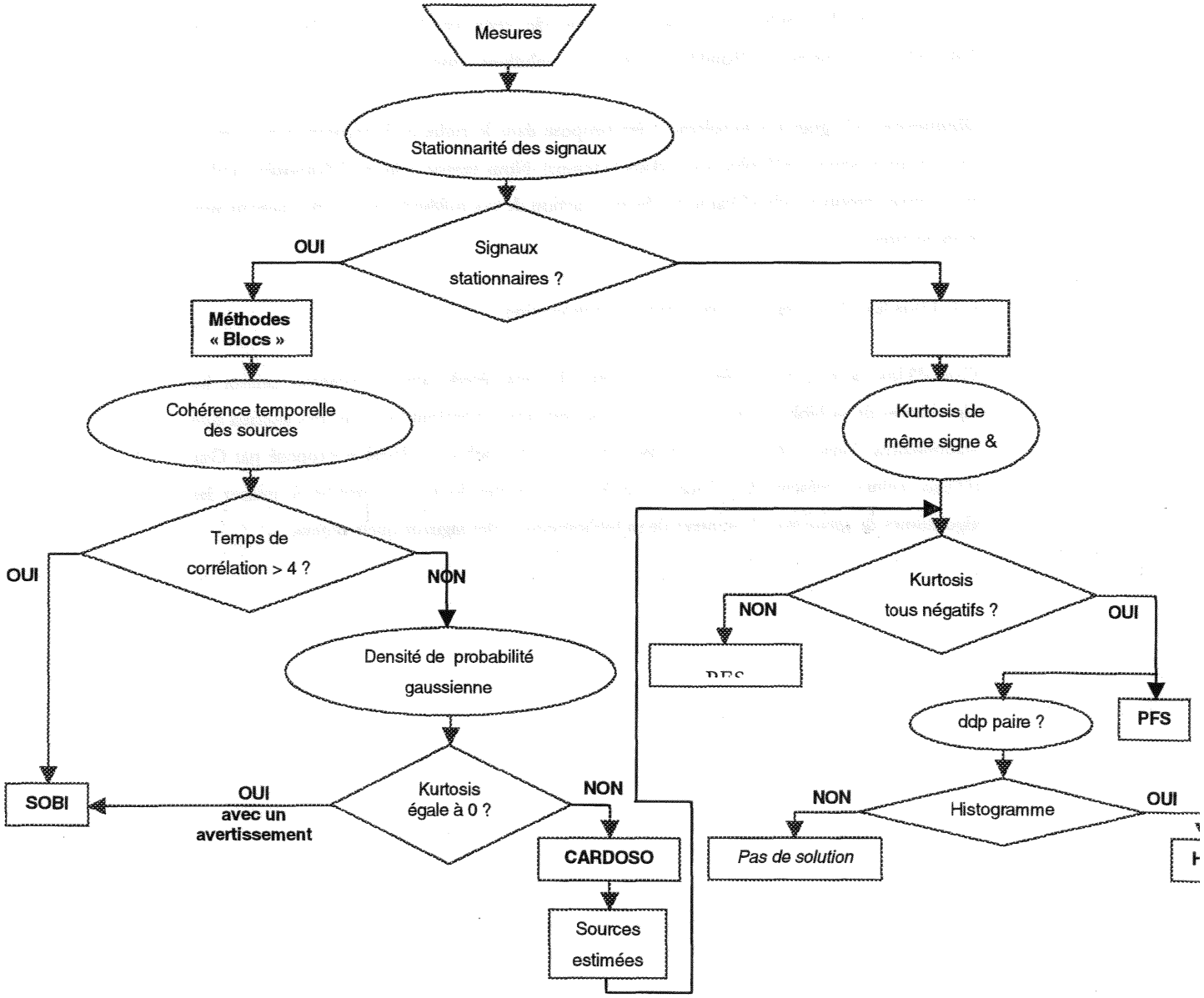


Fig. 35 : La démarche générale de choix d'un algorithme de séparation de sources

Ce schéma descriptif de la démarche que doit suivre l'utilisateur potentiel d'une des méthodes de séparation de sources de la bibliothèque de programme de la DRD a été établi suite aux entretiens successifs que j'ai pu avoir Guy d'Urso.

D. Conclusion

Cette présentation de la séparation de sources, inspirée du rapport HP-21/95/038/A de Guy d'Urso, se veut être une présentation généraliste et synthétique de la séparation de sources au travers de la présentation des algorithmes qui composent la bibliothèque de programmes développée à la DRD d'EDF.

En outre, il décrit de façon simple – ou du moins, aussi simplement que possible - les démarches de choix d'un algorithme en fonction de son contexte d'utilisation (type et caractéristiques des signaux à séparer), démarches qui sont le fruit de l'expertise de Guy d'Urso.

Cependant, la lecture de ce document - qui s'adresse à des non-initiés - nécessitera d'être complétée – pour les spécialistes du domaine - par celle du rapport de Guy d'Urso qui décrit précisément chacun des algorithmes utilisés et illustre sa démarche par la description d'un cas concret.

II. Utilisation par EDF des méthodes de séparation de sources pour la surveillance d'installations

EDF s'est intéressé aux techniques de séparation de sources (mélanges instantanés) afin d'optimiser la surveillance de ses installations. Trois des applications industrielles qui ont été développées par leurs experts en traitement du signal sont présentées dans cette partie.

La première application concerne les réacteurs nucléaires : ceux-ci sont composés de différents éléments qui vibrent et se déforment. Comme nous le verrons, pour une bonne surveillance, il convient de séparer tout particulièrement la signature vibratoire de l'écran thermique et celle du panier de coeur, signatures caractérisées par des modes vibratoires dans le domaine fréquentiel. Les experts de la DRD proposent l'application de deux méthodes de séparation de sources qui permettent d'estimer la densité spectrale de chaque source vibratoire.

En dépit de leur immobilité apparente, les barrages se meuvent au cours de leur exploitation. L'auscultation des ouvrages vise à se prononcer sur leur état de santé. Pour ce faire, il faut distinguer les déplacements du barrage consécutifs aux sollicitations de ceux qui sont symptomatiques d'une dégradation de l'ouvrage.

Enfin, dans le cadre du ***CND*** (Contrôle Non Destructif) réalisé sur les tubes de générateurs de vapeur des centrales nucléaires, l'objectif est de détecter l'apparition de fissures, en surface et en profondeur, dans ces tubes. Il a été proposé une approche multicapteurs pour la restauration des signaux.

A. Introduction

Dans le cadre de la séparation de sources, on dispose de signaux observés à la sortie de capteurs, qui reflètent un mélange des réponses d'un processus à des sollicitations distinctes ou sources. Comme nous l'avons déjà mentionné, la séparation de sources vise à retrouver, le plus fidèlement possible, les différentes sollicitations, uniquement à partir des signaux observés. Cette séparation s'effectue sans connaissance a priori, ni sur le processus, ni sur les signaux sources, mais utilise les propriétés statistiques des sources qui doivent être indépendantes.

Ces techniques multicapteurs sont déjà utilisées en géophysique ou en télécommunication, mais elles n'ont pas encore pénétré les systèmes de surveillance industriels. Toutefois, comme les applications présentées nous donneront l'occasion de l'observer, elles peuvent résoudre certains problèmes réels ardus dans lesquels les excitations du système ne sont pas accessibles.

B. Application à la surveillance de réacteurs nucléaires 900 MW CP0

B.1. Problématique

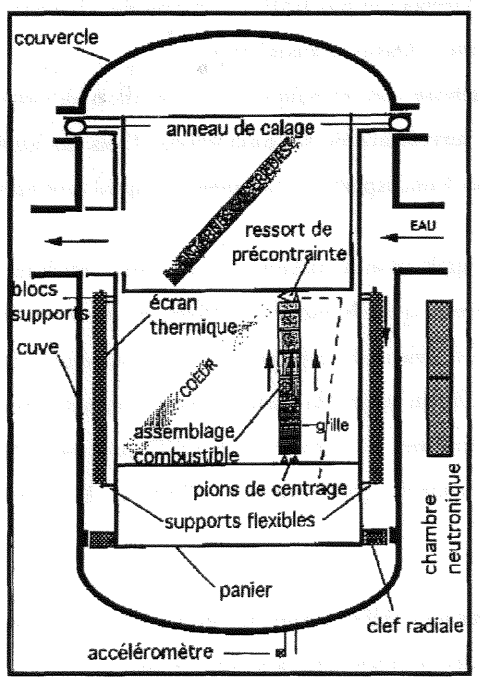


Fig. 36 : Schéma d'un réacteur 900 MW CP0

Un réacteur nucléaire est composé de différents éléments qui vibrent et se déforment. Pour une bonne surveillance, il convient tout particulièrement de séparer la signature vibratoire de l'écran thermique et celle du panier de cœur, signatures qui sont caractérisées par des modes vibratoires dans le domaine fréquentiel. Dans le cas idéal (figure suivante, repère a), tous les

modes vibratoires sont repérables, en particulier le mode 1 (aux environs de 7,5Hz) relatif au mouvement pendulaire du panier de cœur, ainsi que le mode 2 (aux environs de 11,5Hz) dû à la déformation de l'écran thermique. Or, dans de nombreux cas, le mode 1, du fait de son amplitude (figure suivante, repère b) ou de son glissement fréquentiel (il peut glisser de 7,5 à 11,5Hz), masque entièrement le mode 2 (cf. figure suivante, repère c).

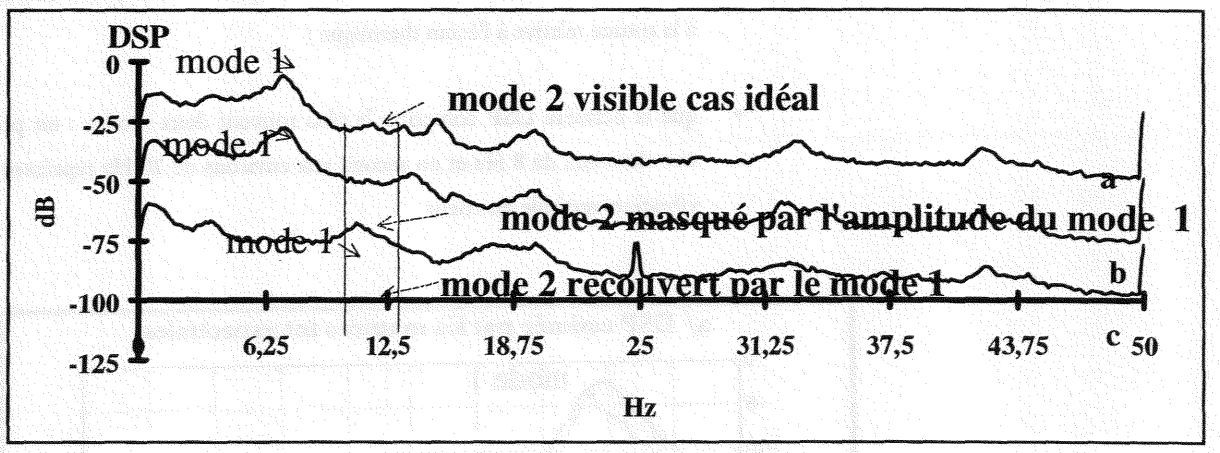


Fig. 37 : ***DSP*** de signaux mesurés par une chambre neutronique

On est en présence de plusieurs sources vibratoires d'origines physiques différentes, donc indépendantes. On dispose de quatre capteurs positionnés à 90° les uns des autres autour du cœur (chambres neutroniques). Chaque capteur délivre un mélange supposé linéaire des diverses sources vibratoires. Le milieu de propagation ne peut être modélisé du fait de sa complexité.

Deux méthodes de séparation de sources ont été appliquées à ce problème : les matrices interspectrales qui permettent d'estimer la densité spectrale de puissance (DSP) de chaque source et l'algorithme SOBI (Second Order Blind Identification) qui effectue une séparation aveugle des signatures temporelles de chaque source.

B.2. Analyse des résultats

Les deux méthodes ont été appliquées avec succès à des données réelles. La première, (figure suivante, repère a), permet d'estimer la densité spectrale de puissance des différentes sources (égales aux valeurs propres). La seconde donne une bonne estimation des sources temporelles séparées. Les DSP des sources séparées sont présentées sur la figure suivante, repère b.

L'examen des DSP révèle (figure suivante):

- que les deux DSP les plus grandes ont une contribution importante à 7,5 Hz (repère a), elles se rapportent donc au balancement du panier du cœur ;
- que la DSP présentant une contribution importante à 11,5 Hz correspond à la source relative à l'écran thermique ;
- que la dernière DSP comporte le plus souvent deux modes : un premier aux alentours de 8 Hz et un second aux environs de 10 Hz représentant la vibration verticale du coeur.

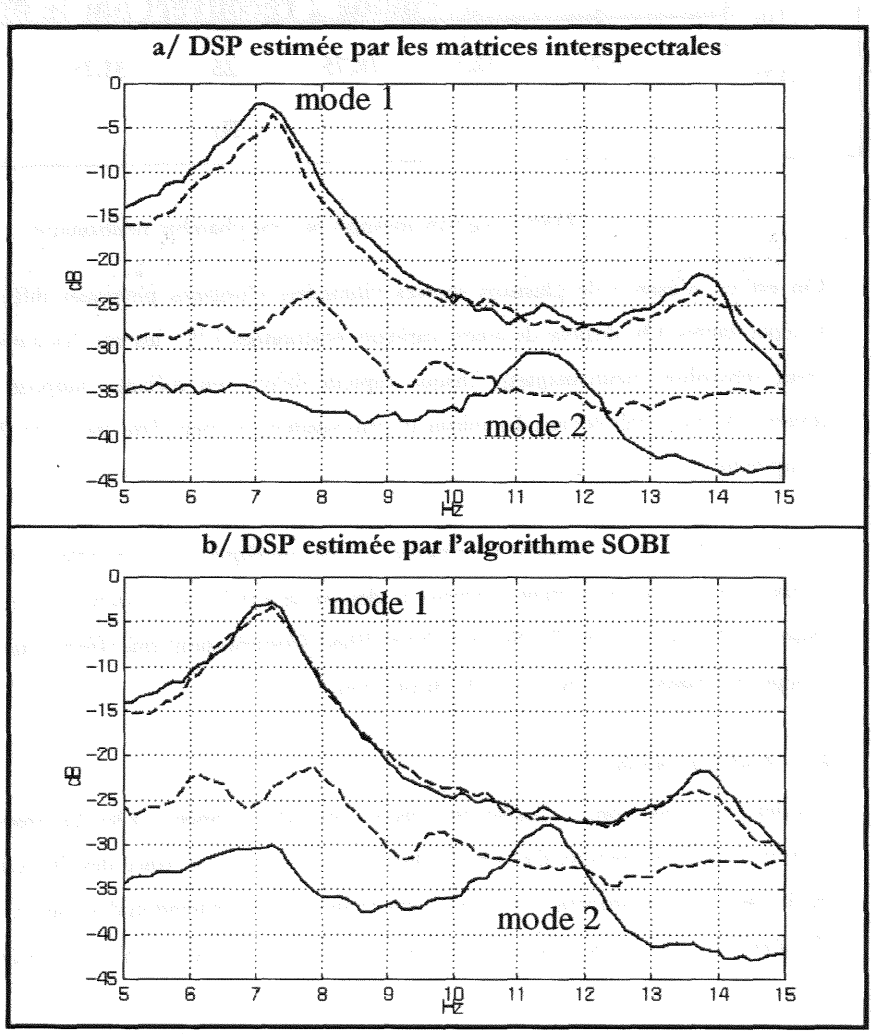


Fig. 38 : Densités spectrales de puissance (DSP) des sources vibratoires estimées.

C. Application à la surveillance des barrages

C.1. Problématique

En dépit de leur immobilité apparente, les barrages se meuvent au cours de leur exploitation. L'auscultation des ouvrages consiste à se prononcer sur leur état de santé. Pour ce faire, il faut distinguer les déplacements du barrage consécutifs aux sollicitations et ceux qui sont symptomatiques d'une dégradation de l'ouvrage.

Les barrages sont équipés de capteurs de déplacement, radial et tangentiel, de capteurs de mesure de débit de fuite, de piézomètres.... Les signaux de surveillance ainsi recueillis contiennent, d'une part l'information sur le vieillissement du barrage (autrement dit l'effet d'usure du temps), d'autre part la réponse du barrage à des sollicitations externes. Ces sollicitations sont de nature :

- mécanique, correspondant à la force de poussée engendrée par la retenue d'eau,
- thermique, associée aux contraintes provoquées par les changements de température.

Il a été supposé que le déplacement du barrage est un mélange additif des réponses du barrage à ces trois phénomènes supposés indépendants et non gaussiens auquel vient s'ajouter un bruit de mesure. Les mesures de déplacement sont peu nombreuses (800 échantillons dans le meilleur des cas) et leur échantillonnage est non régulier (pouvant varier de 3 à 20 jours) et non uniformément réparti.

L'étude menée au sein du Département Surveillance, Diagnostic, Maintenance de la Division Recherche & Développement d'EDF a porté sur les mesures de déplacement.

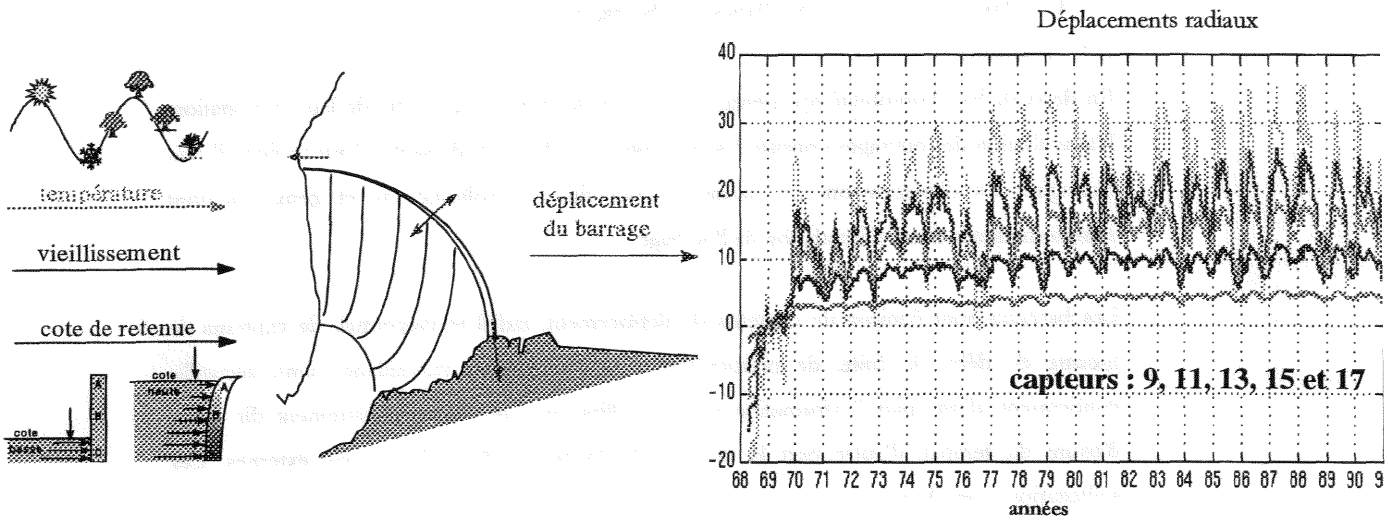


Fig. 39 : Contributions au déplacement d'un barrage

C.2. Méthodes utilisées

Les sources étant non gaussiennes, les spécialistes auraient pu songer à mettre en oeuvre l'un des algorithmes fondés sur des statistiques d'ordre supérieur [6-7], mais ne disposant que d'un faible nombre de points de mesures (de 150 à 800 selon les essais), ils ont dû obter pour une autre méthode : une bonne estimation des matrices de corrélation nécessitant moins d'échantillons que l'estimation de cumulants d'ordre supérieur, ils ont choisi d'utiliser l'algorithme de séparation de sources SOBI [3-4] afin de retrouver les différents contributions à partir des signaux observés sur le réseau de capteurs.

Il faut insister sur le fait que l'hypothèse de mélange instantané permet de d'utiliser les méthodes classiques de séparation de sources, même pour des signaux échantillonnés non régulièrement. SOBI exploite la propriété d'indépendance entre sources qui se traduit techniquement par une structure diagonale des matrices de corrélation. L'estimation de matrices de corrélation pour un échantillonnage irrégulier est certainement biaisée mais leur structure diagonale est conservée. L'algorithme garde donc son efficacité.

C.3. Application au déplacement d'un barrage en béton

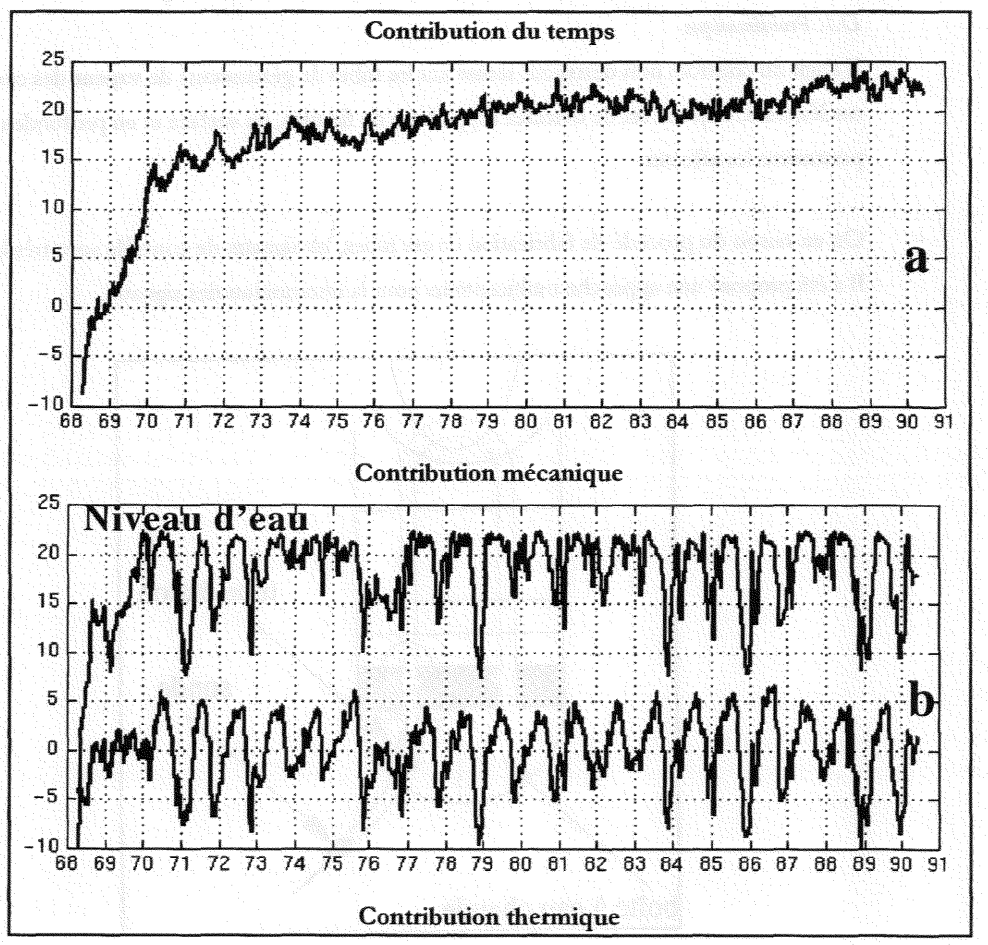
L'utilisation de cinq capteurs (placés sur une même verticale du barrage) permet d'extraire les trois effets, ce qui constitue un bon résultat.

Des mesures journalières de température et de côte de retenue, disponibles pour ce barrage, permettent de valider les résultats obtenus.

La figure suivante présente une visualisation des contributions au regard de la cote de retenue de la température :

- repère a : déplacement dû au vieillissement ;
- repère b en haut : cote de retenue,
  en bas : déplacement dû à la contribution mécanique ;
- repère c en haut : température,
  en bas : déplacement dû à la contribution thermique.

Ces tracés font clairement apparaître un lien de causalité entre les excitations (température et côte de retenue) et leurs contributions sur le barrage : déplacement mécanique et thermique. De plus, l'effet du temps (vieillissement) permet de suivre l'évolution du barrage dans son déplacement vers l'aval.



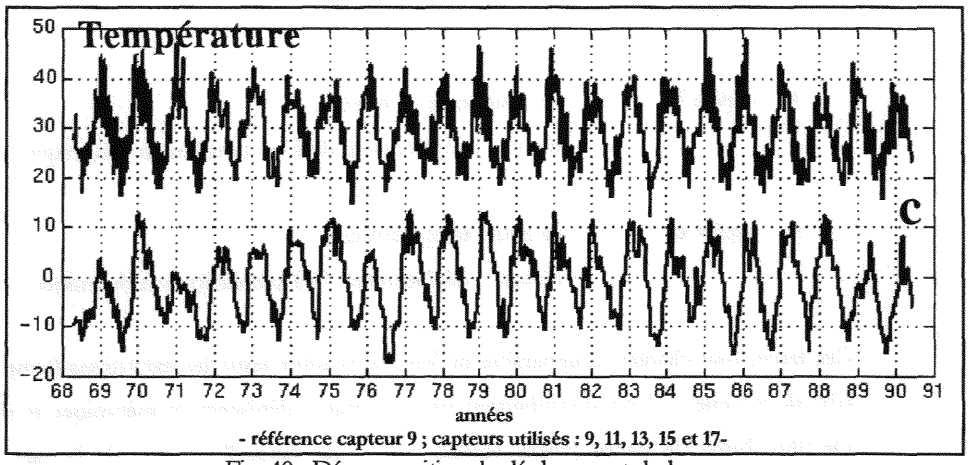


Fig. 40 : Décomposition du déplacement du barrage

D. Application au contrôle des tubes de générateurs de vapeur

D.1. Problématique

Il s'agit du contrôle non destructif réalisé sur les tubes de générateurs de vapeur des centrales nucléaires. L'objectif est de détecter l'apparition de fissures, en surface et en profondeur, qui pourraient les affecter.

Or, en raison du procédé de fabrication de ces tubes, les signaux de contrôle sont très bruités. Il a été proposé une approche multicapteurs pour la restauration des signaux.

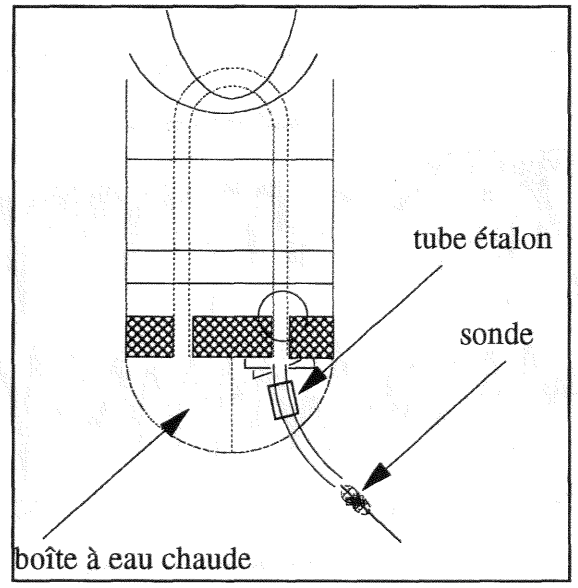


Fig. 41 : Schéma d'un générateur de vapeur

Les tubes sont contrôlés à l'aide de sondes à courants de Foucault. Ces sondes sont sensibles aux irrégularités du matériau et décèlent la présence de défauts qui altèrent la régularité d'un

tube. Or, la surface des tubes, fabriqués par laminage à froid, présentent des ondulations - irrégularités normales - qui génèrent un bruit dit de laminage pouvant perturber le contrôle en masquant les défauts. Il est impératif d'éliminer ce bruit pour rendre le contrôle possible.

Plusieurs voies de mesures complexes sont disponibles simultanément. Parmi les méthodes étudiées jusqu'ici, les plus avancées, en particulier le filtrage adaptatif, reposent sur l'utilisation d'une référence de bruit. En pratique, même si certaines voies contiennent majoritairement du bruit, on ne dispose pas d'une référence de bruit seul, ce qui limite les performances de cette catégorie de méthodes.

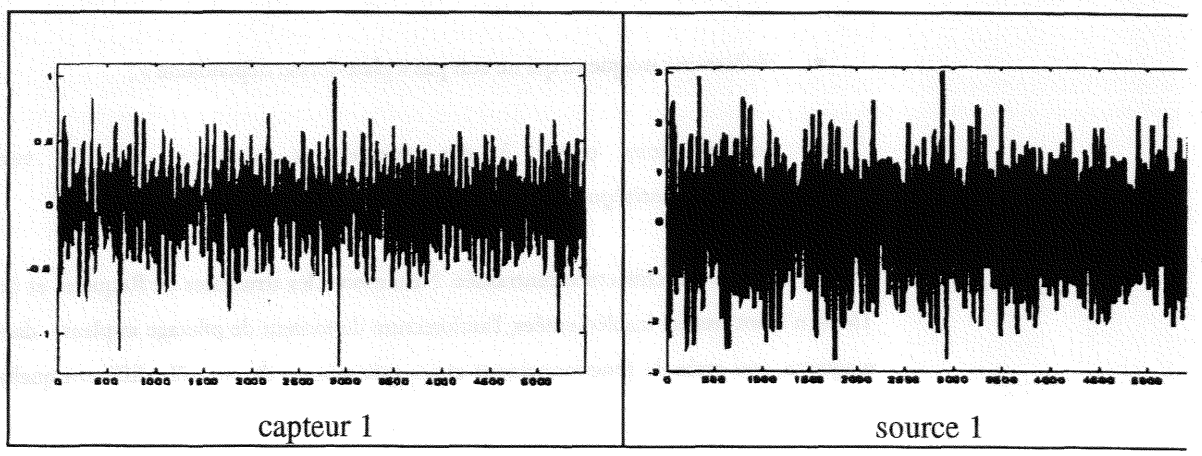
Les experts de la **DRD** ont donc proposé l'utilisation des méthodes de séparation de sources. On considère que plusieurs types d'irrégularités (défaut, bruit de laminage) sont présents dans les tubes et se mélangent au niveau des mesures. On peut assimiler ces irrégularités à des sources excitatrices et supposer qu'elles sont statistiquement indépendantes les unes des autres. Le traitement multi-voies, grâce aux méthodes de séparation, permet de distinguer les contributions des défauts et du bruit de laminage du signal de contrôle.

D.2. Algorithmes de séparation

Il ne pourra être question ici que de l'algorithme de Cardoso (fondé sur des statistiques d'ordre quatre).

C.3. Résultats expérimentaux

Les résultats montrent une amélioration du rapport signal sur bruit. Certains défauts cachés peuvent ainsi être mis en évidence. La figure suivante est une illustration de cet effet. Deux voies réelles Xf1 et Xf2 ont été utilisées comme entrées (à gauche). Le traitement réalise la séparation entre le signal utile et le bruit (à droite). Le défaut qui était invisible dans les signaux bruts a été mis en évidence.



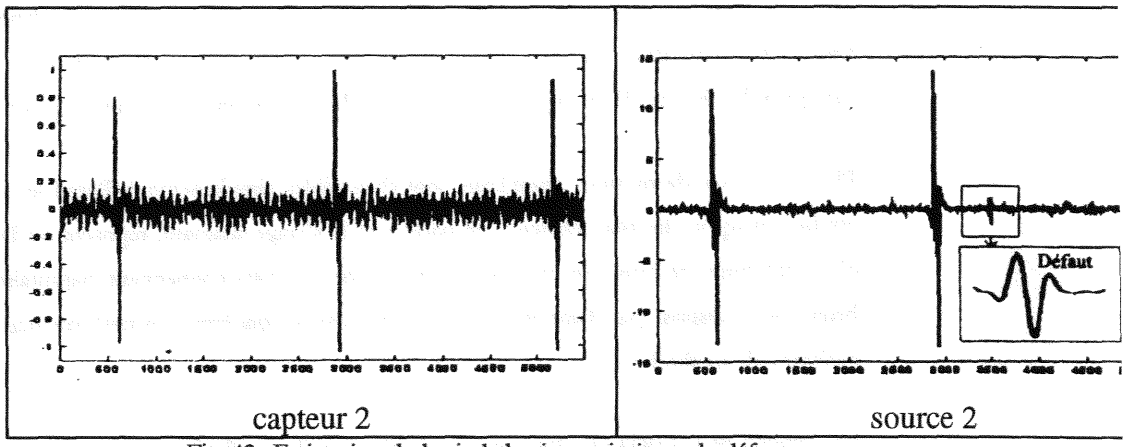


Fig. 42 : Estimation du bruit de laminage ainsi que du défaut

En conclusion, il est possible d'éliminer le bruit de laminage par séparation de sources. Le rapport signal sur bruit est nettement amélioré par ce traitement et les défauts de faibles amplitudes peuvent être rendus visibles.

E. Conséquences

L'étude autour de l'utilisation de ces méthodes de séparation de sources se poursuit actuellement au sein de la Division Recherche & Développement d'EDF sur la comparaison des algorithmes et sur l'évaluation de leurs performances, et c'est justement cet aspect qui a motivé la réalisation de notre maquette comme moyen de choisir la méthode la mieux adaptée au problème en cours de traitement et ce en utilisant les connaissances formalisées par les experts.

III. Description de notre maquette

F. Architecture et interface

A.1. Introduction

La maquette a été réalisée en **JAVA** (sous **Windows NT4**) et ce pour les raisons suivantes : nous souhaitions :

- réaliser une maquette qui ne soit pas « plate-forme dépendante » ;
- pouvoir nous ouvrir à des perspectives réseaux : extension vers une ***« Programathèque Distribuée ».***

Celle-ci utilise une architecture centralisée. Néanmoins, les structures de Requêtes et de Méta-Tâche n'ayant pas été implémentées, l'architecture du moteur de pilotage implanté dans notre maquette, son cycle de fonctionnement et ses mécanismes de contrôle diffèrent quelque peu

de ceux présentés ci-avant (chapitre 3, section II, partie A), comme en témoigne les schémas suivants.

Enfin, signalons que la bibliothèque de programmes que notre maquette pilote est uniquement constituée de modules de séparation de sources développés sous **Matlab** et appelés par notre moteur **JAVA** via une procédure écrite en **C**.

A.2. Architecture du moteur implanté

Le moteur de pilotage doit lui aussi fournir plusieurs fonctionnalités. Il doit permettre d'une part le choix des tâches à exécuter, d'autre part l'exploration des stratégies associées, et, enfin, le contrôle de leur exécution.

Cependant, comme celui-ci ne manipule que des tâches, son architecture, déduite de celle précédemment présentée, se trouve considérablement allégée puisque l'on n'en conserve que le Contrôleur d'Exécution sans son Paramètreur (pour explorer les stratégies des tâches complexes et pour contrôler les résultats de leurs exécutions) et l'Exécuteur de Tâches (pour l'exécution des tâches primitives).

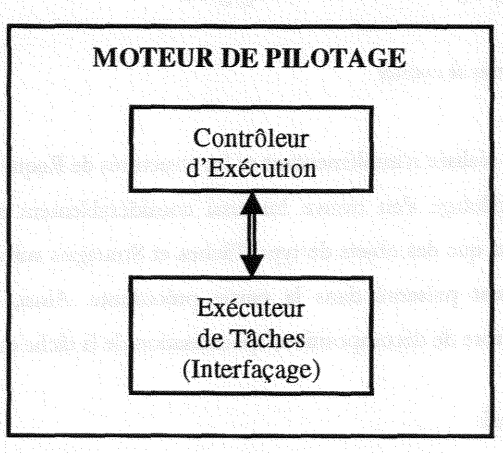


Fig. 43 : Architecture fonctionnelle du moteur de pilotage utilisé

A.3. Rôle des différents composants

Le Contrôleur d'Exécution

Le Contrôleur d'Exécution a pour rôle d'exécuter la Tâche qui lui a été transmise. Si cette Tâche est une tâche primitive – i.e. directement associée à une méthode exécutable –, il l'envoie à l'Exécuteur de Tâches pour exécution. Si, au contraire, cette Tâche est une tâche complexe, le Contrôleur d'Exécution aura pour rôle l'exploration de sa stratégie associée, faisant appel, au besoin - c'est-à-dire à chaque fois que dans celle-ci, une tâche nécessitera

l'évaluation de sa Tâche de précondition - à l'Exécuteur de Tâches pour tester, par l'exécution, les Tâches de précondition.

Remarque : Signalons à cette occasion que choisir une tâche parmi un ensemble – lors d'un choix système - revient à retenir la première tâche spécifiée dans la stratégie de choix dont la précondition est vérifiée, c'est-à-dire dont l'exécution se solde par un succès.

L'Exécuteur de Tâches

Il a la charge de lancer et de surveiller l'exécution des méthodes - i.e. tâches primitives -, s'affranchissant au passage des problèmes d'interfaçage : il doit pouvoir lancer l'exécution d'une méthode interne ou externe, et ce avec les bons arguments. Une fois l'exécution achevée, il renvoie au contrôleur d'exécution le résultat de l'exécution (un booléen permettant de signaler le succès ou l'échec de l'exécution et, éventuellement, les données de sortie produites).

Remarque : L'Exécuteur de Tâches joue ici un rôle extrêmement réduit puisqu'il ne fait que lancer l'exécution du code associé à la tâche qu'il a pour rôle d'exécuter. Signalons à cette occasion, que dans le cadre de notre maquette, les modules **Matlab** manipulés le sont via des appels à des programmes écrits en langage **C**.

A.4. Mécanismes de contrôle

Introduction

La maquette réalisée n'implémentant ni les structures de Requêtes, ni celles de Méta-Tâches, le moteur de pilotage s'en trouve lui-aussi considérablement simplifié. En effet, celui-ci ne manipule plus que des objets de type Tâches et Stratégies suivant un cycle dérivé du cycle de fonctionnement présenté dans la partie précédente. Ainsi, le moteur actuel se contente d'explorer l'arbre de décomposition/spécialisation de la tâche qu'il doit exécuter.

Boucle de contrôle

L'algorithme de contrôle implémenté est donc une simplification du précédent algorithme de contrôle :

1. Recherche & ordonnancement de toutes les Tâches en attente ;

   TANT QU'il reste des Tâches non traitées :

   2. Choix d'une Tâche à traiter (la plus ancienne de la file d'attente) ;

      a) Paramètrisation de la Tâche (initialisation/adaptation de ses paramètres) ;

      b) « Exécution » de la Tâche ;

      SI la Tâche est primitive **ALORS** exécution directe du Code/Module associé ;

      SINON décomposition en sous-tâches par le biais de sa Stratégie associée ;

   3. Evaluation des résultats obtenus ;

Remarques :

1. l'évaluation des résultats dont il est question ici est complètement binaire : les résultats produits lors de cette exécution ne sont en fait nullement évaluée en terme de qualité ; le système se contente de vérifier que l'exécution a été possible et complète ;

2. lorsqu'une tâche a une précondition – c'est-à-dire une tâche lui servant de précondition –, celle-ci est exécutée avant sa tâche englobante et ça n'est que si celle-ci se solde par une réussite que la tâche englobante est exécutée. Ainsi, lors d'un choix système, chacune des tâches en compétition est associée à sa tâche de précondition et c'est la première – dans la liste des tâches associées à ce choix – dont la précondition est vérifiée qui sera choisie pour être exécutée.

B. Instanciation du modèle de tâches

B.1. Rappels

Le problème traité est celui de la « Séparation de Sources » ; nous ne reviendrons pas dans cette partie sur la présentation du problème physique, nous nous contenterons néanmoins de rappeler le graphe de décomposition/spécialisation que nous avons utilisé dans notre maquette.

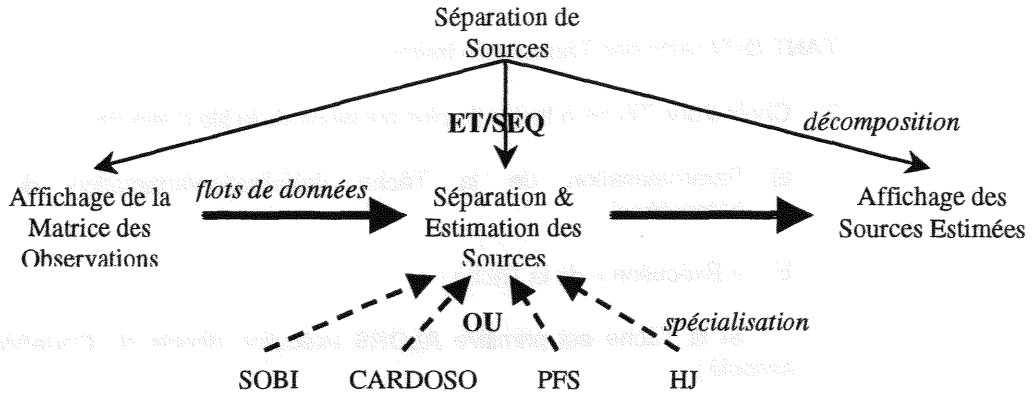


Fig. 44 : Graphe de décomposition/spécialisation utilisé pour la séparation de sources

B.2. Présentation des différentes tâches

Dans cette partie, nous allons succinctement présenter le découpage en Tâches que nous avons utilisé ; pour ce faire, nous allons opter pour une approche descendante : nous partirons de la Tâche générique ***« Séparation de Sources »*** pour nous intéresser ensuite à ses sous-tâches et récursivement.

T0 : « Séparation de Sources »

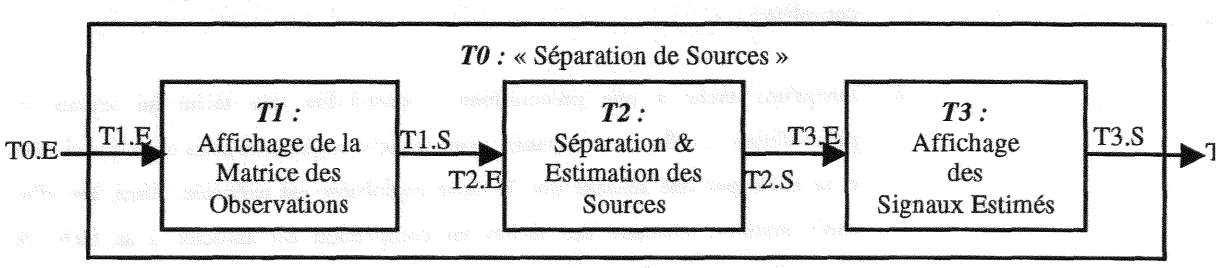


Fig. 45 : Schéma-bloc de la tâche « Séparation de Sources »

Flots de données : T1.E=T0.E ; T2.E=T1.S ; T3.E=T2.S ; T0.S=T3.S

Stratégie associée : (Seq T1 T2 T3)

T1 : « Visualisation Puissance »

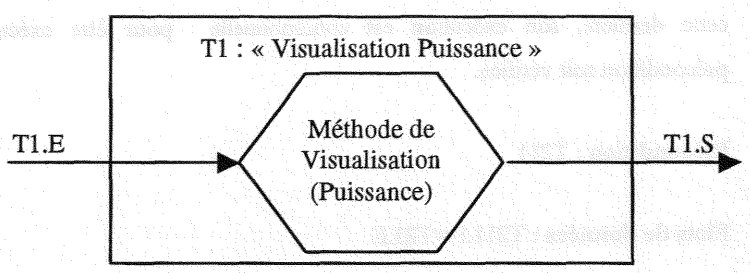


Fig. 46 : Schéma-bloc de la tâche « Visualisation Puissance »

Stratégie associée : T1

T2 : « Séparation »

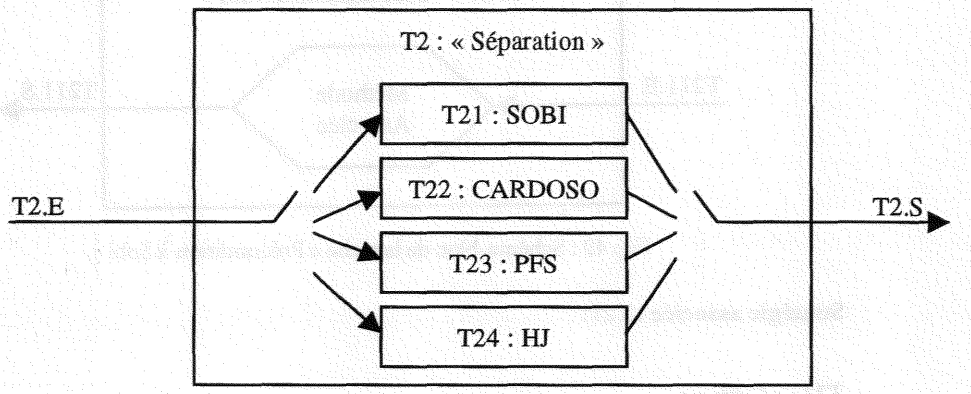


Fig. 47 : Schéma-bloc de la tâche « Séparation »

Flots de données : T21.E=T2.E ; T22.E=T2.E ; T23.E=T2.E ; T24.E=T2.E ; T22.S=T21.S ; T23.S=T21.S ; T24.S=T21.S ; T2.S=T21.S

Stratégie associée : (Csy T21 T22 T23 T24)

T21 : « Sobi »

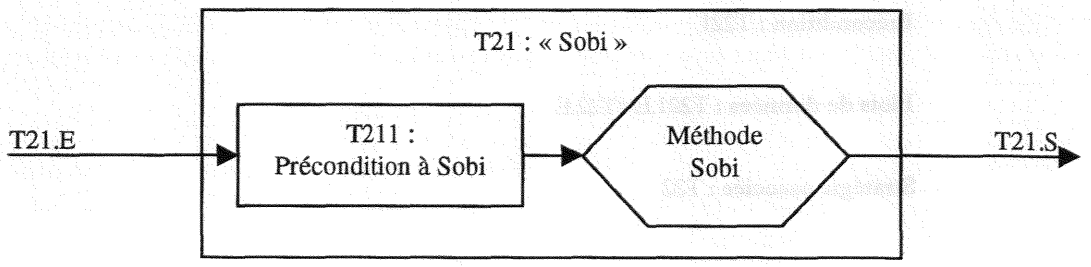


Fig. 48 : Schéma-bloc de la tâche « Sobi »

La Tâche T21 est, comme T1, directement associée à une méthode, mais, contrairement à cette dernière, son exécution est conditionnelle : pour être exécutée, il faut que sa précondition soit vérifiée.

Précondition : T211

Flots de données : T211.E=T21.E

Stratégie associée : T21

T211 : « Précondition à Sobi »

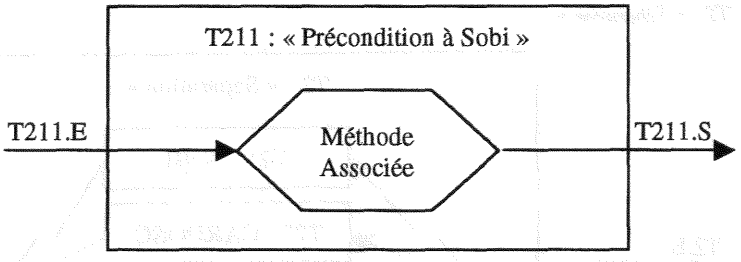


Fig. 49 : Schéma-bloc de la tâche « Précondition à Sobi »

Stratégie associée : T211

T22 : « Cardoso »

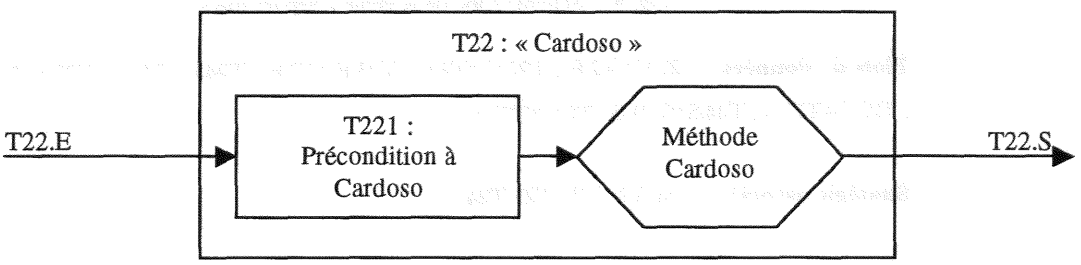


Fig. 50 : Schéma-bloc de la tâche « Cardoso »

Précondition : T221

Flots de données : T221.E=T22.E

Stratégie associée : T22

S.C.D. - U.H.P.
NANCY 1
SCIENCES
BIBLIOTHÈQUE

T221 : « Précondition à Cardoso »

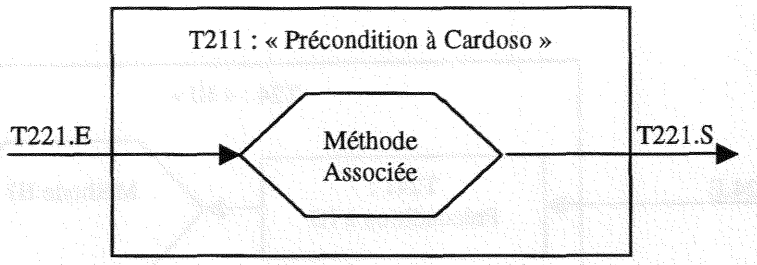


Fig. 51 : Schéma-bloc de la tâche « Précondition à Cardoso »

Stratégie associée : T221

T23 : « PFS »

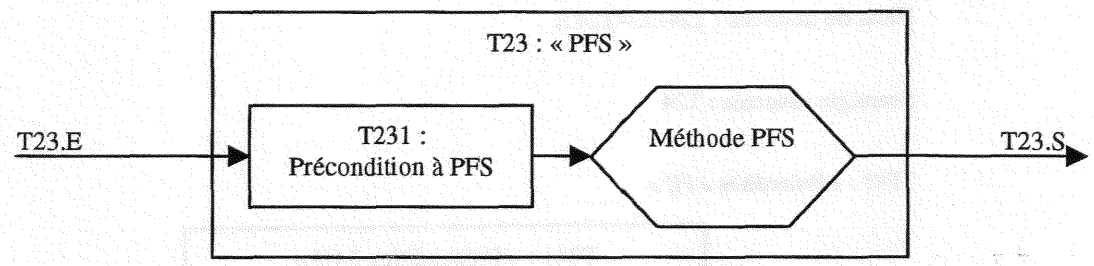


Fig. 52 : Schéma-bloc de la tâche « PFS »

Précondition : T231

Flots de données : T231.E=T23.E

Stratégie associée : T23

T231 : « Précondition à PFS »

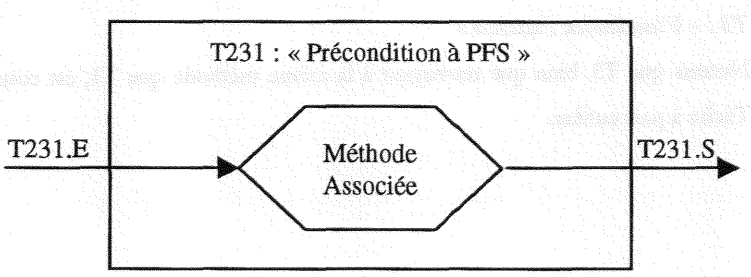


Fig. 53 : Schéma-bloc de la tâche « Précondition à PFS »

Stratégie associée : T231

T24 : « HJ »

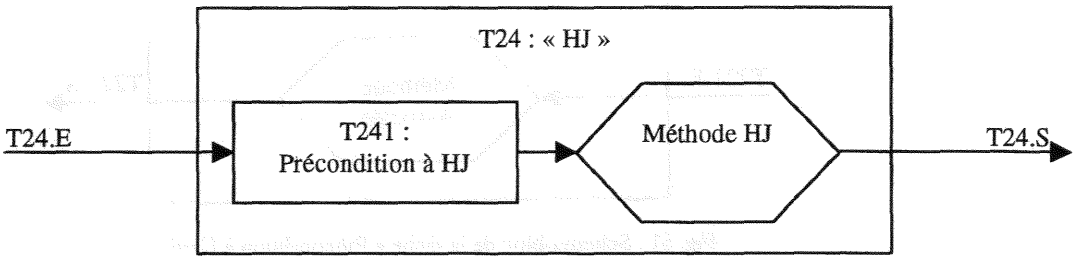


Fig. 54 : Schéma-bloc de la tâche « HJ »

Précondition : T241

Flots de données : T241.E=T24.E

Stratégie associée : T24

T241 : « Précondition à HJ »

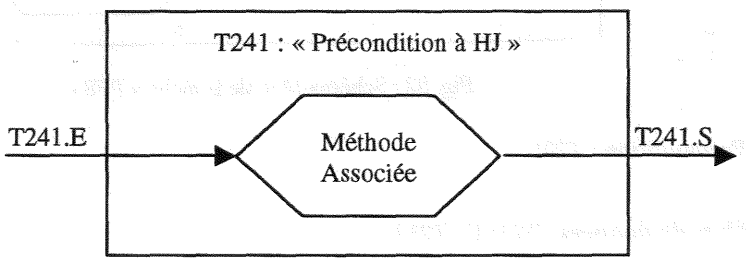


Fig. 55 : Schéma-bloc de la tâche « Précondition à HJ »

Stratégie associée : T241

T3 : « Visualisation Puissance »

Notons que T3, bien que renvoyant à la même méthode que T1, est considérée comme une Tâche à part entière.

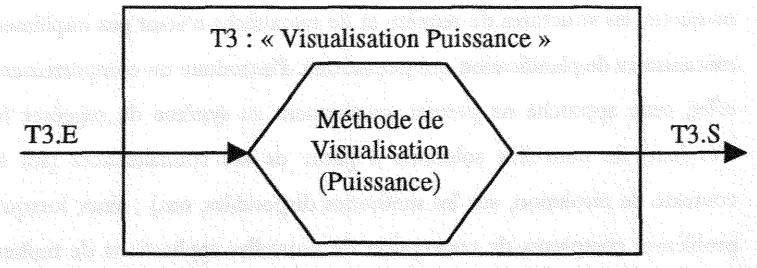


Fig. 56 : Schéma-bloc de la tâche « Visualisation Puissance »

Stratégie associée : T3

B.3. Ordre de déclaration des différentes tâches

Contrairement à la précédente présentation, les Tâches doivent être déclarées dans l'ordre ascendant, c'est-à-dire qu'on commence par définir les « briques » de base avant de définir les Tâches plus complexes qui les utilisent.

Ainsi l'ordre de déclaration des Tâches précédentes devra être le suivant : T1 ; T211 ; T21 ; T221 ; T22 ; T231 ; T23 ; T241 ; T24 ; T2 ; T3 ; T0.

IV. Résultats et limites

A. Les points forts de cette réalisation

Dans le cadre de cette maquette, nous avons pu mettre à l'épreuve un certain nombre de concepts que nous avions proposés. Ainsi, nous sommes convaincu du bien-fondé de notre approche par Modèle de Tâches qui permet une description abstraite, de haut-niveau, des programmes d'une bibliothèque et des traitements qu'ils opèrent, ce qui facilite l'aide à la conception de nouveaux logiciels et la réutilisabilité des composants pré-existants (génie logiciel).

Cette approche est également nécessaire pour permettre, non seulement la description, mais aussi la manipulation et la compréhension d'algorithmes, car cela permet au système de raisonner sur les tâches, sur leurs caractéristiques et sur les objets qu'elles manipulent.

B. Les limites de cette réalisation

Dès qu'une approche descriptive et descendante (décomposition du problème initial en sous-problèmes plus simples) peut être adoptée – comme c'est le cas lorsque l'on cherche seulement à combiner des traitements en boîte-noire –, le Modèle de Tâches et le Moteur implémentés dans cette maquette semblent répondre au cahier des charges ; cependant, dès que l'on cherche à enchaîner de façon non déterministe un certain nombre de traitements en fonction des buts poursuivis et du contexte d'exécution, cette approche par Modèle de Tâche et par décomposition/spécialisation semble présenter ses limites – rappelons que dans cette

maquette, les structures de requête et de méta-tâche n'étant pas implémentées, il n'y a pas de mécanismes de planification qui permettrait d'introduire un comportement opportuniste -. En effet, cette approche ne permet aucunement au système de suggérer (i.e. de construire et d'évaluer) de nouvelles solutions à partir de ses connaissances (sur le problème, sur le contexte de résolution, sur les méthodes disponibles, etc.) ; ainsi, lorsqu'on s'intéresse à des problèmes complexes de conception de nouvelles applications de traitement du signal ou à des problèmes d'interprétation ou de reconnaissance de formes, cette approche reste encore trop limitée du fait de son fort caractère centralisé et de son fonctionnement purement descendant.

C. Conséquences sur notre modèle

C.1. Extension de notre modèle de tâches

Cette première réalisation construite autour du problème de la séparation de source avait pour but de faire apparaître l'intérêt d'une approche basée sur le pilotage de programmes et l'aptitude de notre modèle à répondre à nos besoins. Ainsi, grâce à celle-ci, nous avons pu déceler un certain nombre de carences dans notre modèle, ce qui nous a amenés à le compléter. Dans ce cadre, nous avons introduit un certain nombre de nouvelles entités :

- les requêtes pour permettre à l'utilisateur ou au système de demander la résolution d'un problème ;

- les méta-tâches qui permettre d'agréger des tâches permettant de réaliser la même action et de préciser les règles de choix qui préside au choix d'une méthode parmi les méthodes disponibles ;

- les entités qui permettent de caractériser et d'encapsuler les objets externe manipulés par les tâches.

C.2. Extension multi-agents de notre moteur de pilotage

De même, ce sont les observations faites sur notre première maquette qui nous ont conduits à considérer une approche mixte de type multi-agents, combinée à notre formalisme par Modèle de Tâches. En effet, la réalisation d'une tâche complexe peut-être vue comme la mise en commun des compétences de différents intervenants (ou agents), chacun sachant ce qu'il est capable de produire en fonction de ce dont il dispose. Ainsi, le formalisme par Modèle de Tâche permettra la description des Tâches à réaliser (buts, entrées/sorties, etc.) et le dialogue entre les différents agents qui, dès lors, pourront coopérer tout en restant guidés par les connaissances de l'utilisateur qui continuera à pouvoir définir des stratégies de résolutions lorsqu'il en aura la compétence.

Les motivations qui sont à l'origine de ce choix sont directement liées aux limites de notre première approche. En effet, le moteur de pilotage centralisé que nous avons implémenté dans notre maquette présentait les limites suivantes :

- il ne permettait que de *« dérouler »* les stratégies pré-établies en fonction du contexte ;
- il ne permettait aucunement :
    - de construire/explorer de nouvelles solutions en fonctions de ressources disponibles (nouvelles méthodes mise à disposition par un autre expert…)
    - d'aborder des problèmes plus complexes d'interprétation de signaux qui nécessitent la prise en compte et l'analyse direct des données disponibles pour construire une interprétation ;
    - de piloter des entités complexes telles que des agents d'interprétation.

Ainsi, les apports d'une approche multi-agents sont nombreux : cela permet d'intégrer les phases de raisonnement ascendantes (i.e. guidées par les données) et les mécanismes de contrôle opportunistes nécessaires à la réalisation de tâches complexes.

Associé au mécanisme de requêtes, cela permet aussi l'émergence de nouvelles solutions en fonction des ressources disponibles.

Enfin, cette extension permet l'introduction d'un certain nombre de concepts qui nous semblent particulièrement prometteurs :

- les tâches réactives : en fonction des données observées, une tâche pourra demander à être activée, ce qui peut permettre la mise en place de mécanismes d'association perception → action ;
- la coopération et l'interaction entre spécialistes qui peuvent s'associer pour produire le résultat attendu ;
- la compétition entre spécialistes qui permet de ne retenir que les agents les plus performants dans le domaine abordé ;
- l'organisation et la ré-organisation de la société de spécialistes de sorte à modifier la stratégie de résolution utilisée dans la résolution en cours.

C.3. Réalisation d'une nouvelle maquette

La maquette développée au cours ce cette étape de réflexion n'implémente qu'une partie des concepts qu'il nous semble indispensable de mettre en œuvre dans le cadre d'un outil d'aide développement. Ainsi, il nous semble intéressant de prévoir la mise en chantier d'une nouvelle maquette permettant de tester notre approche dans sa totalité.

Chapitre 4

PERSPECTIVES & CONCLUSIONS

I. Conclusions

La capitalisation des savoir-faire est un des enjeux les plus important de cette fin de Xxème siècle ; ainsi dans la majorité des grandes entreprises, des projets ont vu ou voient le jour autour de cette problématique : il s'agit de se doter d'outils efficaces permettant la bonne mise en œuvre du concept de réutilisation.

C'est dans ce cadre générique que s'inscrit notre travail : en effet, le travail réalisé dans le cadre de cette thèse avait pour objectif de définir un outil permettant la capitalisation de méthodes logicielles et des connaissances associées (conditions d'utilisation, de validité, etc.) par le biais de la définition d'un ***Environnement d'Aide à la Conception et au Maquettage d'Applications de Traitements du Signal et d'Interprétation de Signaux.***

Après avoir étudié les travaux issus de communautés aussi variées que celles de l'Intelligence Artificielle, du Traitement du Signal, des Mathématiques ou du Génie Logiciel, nous avons choisi de partir d'une approche de type ***Environnement d'Aide à la Résolution de Problème*** afin de construire notre ***Bibliothèque de Programmes « Intelligente »*** basée sur un ***Modèle de Tâches*** et sur un ***Moteur de Pilotage*** adapté aux besoins que nous avions préalablement mis en évidence. Le modèle de tâches et l'architecture de pilotage multi-agents que nous avons défini se veulent génériques et propose une solution intéressante, parce qu'efficace, modulaire et évolutive, au problème posé.

Notre modèle de tâches a été conçu de sorte à permettre la formalisation et la manipulation des connaissances expertes de natures variées (symbolique et numériques) ; la structure de requête a permis d'ajouter un degré de liberté dans la description du mécanisme de résolution d'une tâche en permettant à son auteur de définir une étape de la résolution non pas par la méthode opérationnelle associée, mais seulement par le libellé de l'action qui devra être exécutée.

Notre moteur de pilotage distribué, quand à lui, permet de piloter des algorithmes complexes (eux-aussi pouvant être de nature symbolique ou numérique) et de mettre en jeu des mécanismes de raisonnement mixtes mêlant des phases ascendantes (i.e. guidées par les données) et des phases descendantes (i.e. guidées par les buts). Cette approche distribuée puise sa justification dans les limites des approches centralisées qui ne permettent que

l'exploration de stratégies plus ou moins figées et pré-établies et qui n'offre aucun moyen de piloter des agents d'interprétations.

II. Perspectives

Dans la droite ligne du travail réalisé dans cette étude, un certain nombre de compléments sont à apporter. En effet, la maquette développée au cours de cette réflexion bien que portant sur une application relativement simple, nous a permis de mettre le doigt sur un certain nombre de problèmes qui nous ont conduit à compléter et à étendre notre approche initiale. Cependant, n'ayant pas eu l'occasion d'implémenter la totalité des concepts qu'il nous semble indispensable de mettre en œuvre dans le cadre d'un outil d'aide au développement, il me semblerait intéressant de consacrer du temps et des moyens au développement d'une maquette implémentant notre modèle dans sa totalité et ce dans le cadre de l'approche multi-agents.

D'autre part, les choix effectués par le système (c.. la fonction ***BestFit*** de notre formalisation) sont pour l'instant basé sur une expertise codifiée sous forme de règles, or une telle expertise n'est pas toujours disponible ou formalisable ; ainsi il nous semble déterminant de nous intéresser à la possibilité de définir de nouveaux mécanismes de choix basés sur d'autres modèles décisionnels et ce en tenant compte d'éventuels mécanismes d'apprentissage qui permettraient de s'affranchir du difficile problème d'acquisition et de formalisation des connaissances expertes.

Enfin, ayant opté pour un modèle et une implémentation modulaire (***« approche objet »***), un certain nombre d'extensions de celui peuvent être envisagées afin de permettre la distribution des connaissances et des méthodes sur plusieurs sîtes distants, la prise en compte de contrainte de ressources (contraintes de temps, de charge cpu, de localisation, etc.).

De façon plus générale, je pense qu'il serait intéressant de *« faire sauter »* les cloisons qui séparent les différentes communautés intéressées par le problème de la capitalisation des méthodes et savoir-faire, et de mettre en chantier un travail commun avec des experts de ces différentes communautés de façon à ce qu'ils puissent mettre en commun leurs aptitudes dans des domaines très différents, mais presque toujours indissociables.

[Audrain 90a]
M. Audrain, F. Charpillet, D. Dobbeni, D. Fohr, Y. Gong, J. P. Haton, M. Katz et T. Mondot. *Report on development system, real-time inference engine and knowledge base as in progress for prototype 1.* Rapport ESPRIT II 2167. COG/D/IR1, 1990.

[Audrain 90b]
M. Audrain et T. Mondot. *Rules of application 1.* Rapport ESPRIT II 2167. COG/W/Ru1, 1990.

[Audrain 92a]
M. Audrain, F. Charpillet, D. Dobbeni, D. Fohr, Y. Gong, P. Knoop, M. Katz et T. Mondot. *Study of the application of AI techniques to EC processing.* Rapport ESPRIT II 2167. COG/D/D7, 1992.

[Audrain 92b]
M. Audrain, F. Charpillet, D. Dobbeni, D. Fohr, Y. Gong, P. Knoop, M. Katz et T. Mondot. *AITRAS user's guide.* Rapport ESPRIT II 2167. COG/D/D12, 1992.

[Basseville 96]
M. Basseville et M-O. Cordier. *Surveillance et diagnostic de systèmes dynamiques : approches complémentaires du traitement de signal et de l'intelligence artificielle.* Projets AS/REPCO, Rapport de recherche n°2861, Avril 1996.

[Breiman 84]
L. Breiman, J. H. Friedman, R.A. Olshen et C. J. Stone. *Classification and regression trees.* Wadsworth Inc., Belmont California, 1984.

[Brooks 91]
R. A. Brooks. *Intelligence without representation.* Artificial Intelligence, vol. 47, pp 139-159, 1991.

[Carver 91]
N. Carver et V.R. Lesser. *A new framework for sensor interpretation; planning to resolve sources of uncertainty.* The Proceedings of the 1991 National Conference on Artificial Intelligence (AAAI-91), pp 724-731, Anaheim, California, July 1991.

[Carver 92]
N. Carver et V.R. Lesser. *The Evolution of Blackboard Control Architectures.* CMPSCI Technical Report 92-71, Octobre 1992.

[Carver 94]
N. Carver et V.R. Lesser. *Blackboard systems for knowledge-based signal understanding.* Symbolic and knowledge-based signal processing, Prentice Hall signal processing series, chapitre 6, 1994.

[Chaillot 93]
M. Chaillot. *Une Architecture de Contrôle Réactif pour la Résolution Coopérative de Problèmes.* Thèse de doctorat de l'Institut National Polytechnique de Grenoble (INPG), Septembre 1993.

[Chandrasekaran 84]
B. Chandrasekaran. *Expert Systems : Matching Techniques to Tasks.* Artificial Intelligence in Business, W. Reitman, eds. , Ablex Publishing, pp 116-132, 1984.

[Chandrasekaran 86]
B. Chandrasekaran. *Generic Tasks in Knowledge-Based Reasoning : High-Level Building Blocks For Expert System Design.* IEEE Expert 1, vol. 3, pp 23-30, 1986.

[Chandrasekaran 89]
B. Chandrasekaran. *Task-structures, Knowledge Acquisition and Learning.* Machine Learning, vol. 4, pp 339-345, 1989.

[Chandrasekaran 92]
B. Chandrasekaran, T. Johnson et J.W. Smith. *Task Structure Analysis for Knowledge Modeling.* Communication of the ACM, vol. 33, n°9, pp 124-136, 1992.

[Chandrasekaran 93]
B. Chandrasekaran et T. Johnson. *Generic Tasks and Task Structures : History, Critique and New Directions.* Second Generation Expert Systems, eds., J.M. David , J.P. Krivine et R. Simmons, Springer-Verlag, pp 239-280, 1993.

[Chandrasekaran 97]
B. Chandrasekaran et J.R. Josephson. *The Ontology of Tasks and Methods.* AAAI 1997 Spring Symposium on Ontological Engineering, Stanford University, CA, March 24-26, 1997.

[Charpillet 97]
F. Charpillet et A. Boyer. *Progress : un modèle d'agent pour la conception de systèmes multi-agents temps-réel.* Actes des Journées Francophones IA distribuée et Systèmes Multi-agents (JFIADSMA), ed. Hermes, 1997.

[Chevrier 93]
V. Chevrier. *Etude et mise en œuvre du paradigme multi-agents : de ATOME à GTMAS.* Thèse de doctorat de l'Université Henri Poincaré, Nancy I, Juin 1993.

[Clément 91]
V. Clément et M. Thonnat. *PROGAL : un système expert en traitements d'images de galaxies.* Huitième Congrés Reconnaissance des Formes et Intelligence Artificielle, Lyon-Villeurbanne, France, 25-29 Novembre 1991.

[Clouard 94]
R. Clouard. *Raisonnement Incrémental et Opportuniste Appliqué à la Construction Dynamique de Plans de Traitement d'Images.* Thèse de doctorat de l'Université de Caen, Février 1994.

[Crampé 94]
I. Crampé. *Sémantique des tâches décomposables et spécialisables.* Projet SHERPA, Rapport de DEA, Juin 1994.

[Darricau 99]
M. Darricau. *Contribution à la réutilisation de connaissances pour l'aide à l'élaboration et à l'évaluation des spécifications de logiciels – Application à la sécurité des systèmes de transports guidés.* Thèse de doctorat de l'Université Paris VI, Janvier 1999.

[Dawant 91]
B. Dawant et B. Jansen. *Coupling numerical and symbolic methods for signal interpretation.* IEEE Transactions on Systems, Man and Cybernetics, pp 115-124, Janvier-Février 1991.

[Delouis 93]
I. Delouis. *LISA : Un Langage Réflexif pour la Modélisation du Contrôle dans les Systèmes à Base de Connaissances – Application à la Planification des Réseaux Electriques.* Thèse de doctorat de l'Université de Paris-Sud, Centre d'Orsay, Paris, Juin 1993.

[Dojat 94]
M. Dojat. *Contribution à la Représentation d'Expertises Médicales Dynamiques : Application en Réanimation Médicale.* Thèse de doctorat de l'Université de Technologie de Compiègne, Novembre 1994.

[d'Urso 96]
G. d'Urso. *Qu'est-ce que la séparation de sources ?.* Rapport interne, EDF HP-21/95/038/A, Janvier 1996.

[d'Urso 98]
G. d'Urso, P. Prieur et C. Vincent. *Premiers apports des méthodes de séparation de sources appliquées à la surveillance des installations d'EDF.* Senlis, 1998.

[Erman 80]
L. Herman, F. Hayes-Roth, V. R. Lesser et R. Reddy. *The HEARSAY-II speech understanding system: integrating knowledge to resolve uncertainty.* Computing surveys, 12:213-253, 1980.

[Ferber 97]
J. Ferber et O. Gutknecht. *Aalaadin : a meta-model for the analysis and design of organizations in multi-agent systems.* Rapport de Recherche, R.R.LIRMM 97189, Décembre 1997.

[Fink 97]
E. Fink. *Statistical Selection Among Problem-Solving Methods.* Research Report CMU-CS-97-101, Carnegie Mellon University, Pittsburgh, January 1997.

[Foisel 96]
R. Foisel, V. Chevrier et J-P. Haton. *Improving global coherence by adaptive organization in a multi-agent system.* Second International Conference On Multi-Agent Systems, Kyoto, Japon, AAAI Press / MIT Press, pp 435, Décembre 1996.

[Foisel 97]
R. Foisel, V. Chevrier et J-P. Haton. *Un modèle pour la réorganisation de systèmes multi-agents.* Actes des journées francophones IA distribuée et systèmes multi-agents, Colle Sur Loup, pp 261-277, Avril 1997.

[Foisel 98]
R. Foisel. *Construire des systèmes multi-agents à partir de schémas d'interactions.* Actes des 6èmes journées francophones IA distribuée et systèmes multi-agents, Abbaye des Prémontrés, Pont-à-Mousson, pp 295-308, Novembre 1998.

[Forrest 89]
S. Forrest et M. Mitchell. *What makes a problem hard for genetic algorithm? Some anomalous results and their explanation.* Machine Learning , 13:285-319, 1993.

[Fujita 96]
S. Fujita et V. R. Lesser. *Centralized Task Distribution in the Presence of Uncertainty and Time Deadlines.* Proceedings of ICAMS, Japan, 1996.

[Gallone 97]
J. M. Gallone. *Résolution de problèmes sous contrainte de resources à l'aide de réseaux neuromimétiques : application à l'ordonnancement.* Thèse de doctorat de l'Université Henri Poincaré, Nancy I, Janvier 1997.

[Garvey 96]
A. J. Garvey. *Design-To-Time Real-Time Scheduling.* Dissertation presented for the degree of Doctor of Philosophy, University of Massachusetts, Amherst, Department of Computer Science, February 1996.

[Gascuel 95]
O. Gascuel et P. Gallinari (Groupe SYMENU). *Méthodes symbolique-numériques de discrimination.* Actes des 5èmes journées nationales, PRC-GDR, Intelligence Artificielles, 1995.

[Glaser 96]
N. Glaser. *Contribution to knowledge acquisition and modelling in a multi-agent framework : the CoMoMAS approach.* Thèse de doctorat de

l'Université Henri Poincaré, Nancy I, Novembre 1996.

[Goldberg 89]
D. E. Goldberg. *Genetic algorithms in search, optimization, and machine learning.* Addison-Wesley publishing company, inc., 1989.

[Gong 90]
Y. Gong et J-P Haton. Towards a general signal interpretation system : signal-to-symbol conversion level. In 10th IEEE International Conference on Pattern Recognition, vol. II, pp 79-84, Atlantic City, USA, IEEE Computer Society, Juin 1990.

[Gong 94]
Y. Gong. *G^{σ} : a Generic Signal-to-Symbol Conversion System for Signal Interpretation.* Technical Report, Mars 1994.

[Gruber 87]
T. Gruber et P. Cohen. *Design for Acquisition : Principles of Knowledge System Design to Facilitate Knowledge Acquisition.* Int. J. Man-Machine Studies, vol. 26, n°2, pp 143-159, 1987.

[Guin 97]
N. Guin. *Reformuler et classer un problème pour le résoudre : l'Architecture SYRCLAD et son application à quatre domaines.* Thèse de doctorat de l'Université Paris VI, 1997.

[Gutknecht 98]
O. Gutknecht et J. Ferber. *Un méta-modèle organisationnel pour l'analyse, la conception et l'exécution de systèmes multi-agents.* Actes des 6èmes journées francophones IA distribuée et systèmes multi-agents, Abbaye des Prémontrés, Pont-à-Mousson, pp 267-280, Novembre 1998.

[Haïk 95]
P. Haïk, M.-L. Deflandre, P. Moigne, S. Muller, A. Philippe et E. Rapisarda. *Le Système GPS.* Communication et Navigation, Plaquette Scientifique AESM 1995, Ecole Supérieure d'Electricité, 1995.

[Haïk 96]
P. Haïk. *Un environnement d'aide à la conception et au maquettage de systèmes d'interprétation de signaux et de diagnostic : définition d'une architecture générique pour l'interprétation de signaux en environnement dynamique.* Mémoire de D.E.A. Informatique, Université de Nancy I, UFR STMIA, Département Informatique, Septembre 1996.

[Haïk 97]
P. Haïk et F. Charpillet. *Définition d'un Modèle de tâches.* Rapport d'avancement, EDF – INRIA Lorraine, Octobre 1997.

[Haïk 98]
P. Haïk, F. Charpillet et A. Joussellin. *Modèles de tâches et bibliothèques de programmes intelligente pour l'interprétation de signaux & le traitement du signal.* Actes de la conférence Complex Systems, Intelligent Systems & Interfaces, Nîmes, Mai 1998.

[Haïk 99]
P. Haïk, F. Charpillet et G. d'Urso. *Un Environnement d'Aide à la Conception et au Maquettage basé sur une Bibliothèques de Programmes « Intelligente ».* Communication en cours de soumission pour RFIA 2000, Paris, 1-3 Février 2000.

[Haton 89a]
J-P. Haton. *Panorama des systèmes multi-agents.* Onzièmes journées francophones sur l'informatique. Architectures avancées pour l'IA.

[Haton 89b]
J-P. Haton et M-C Haton. *L'Intelligence Artificielle.* Que sais-je?, Presse universitaire de France 1989.

[Haton 91]
J-P. Haton, N. Bouzid, F. Charpillet, M-C. Haton, B. Lâastri, H. Lâastri, P. Marquis, T. Mondot, A. Napoli. *Le raisonnement en Intelligence Artificielle.* Inter-edition 1991.

[Hewitt 73]
C. Hewitt, P. Bishop et R. Steiger. *A universal modular actor formalism for Artificial Intelligence.* Proceedings of the 3rd IJCAI, pp 235-245, Standford, California, 1973.

[Hudlická 87]
E. Hudlická et V.R. Lesser. *Modeling and diagnosing problem-solving system behavior.* IEEE Transactions on Systems, Man and Cybernetics, Special issue on diagnostic reasoning, vol. 17, n°3, pp 407-419, Mai-Juin 1987.

[Jean-Marie 92]
F. Jean-Marie et J. Willamowski. *Modélisation de dialogues et de tâches pour le développement de systèmes coopératifs en calcul scientifique.* Projet SCARP, Le modèle de tâches implémenté dans SCARP, Novembre 1992.

[Joussellin 97]
A. Joussellin, P. Haïk et F. Charpillet. *Etude préliminaire pour la définition d'un environnement de diagnostic associant des méthodes numériques et symboliques.* Rapport interne, EDF HP-21/97/028/A, Décembre 1997.

[Kaelbling 98]
Leslie Pack Kaelbling, Michael L. Littman et Anthony R. Cassandra. *Planning and Acting in Partially Observable Stochastic Domains.* Artificial Intelligence, vol. 101, 1998.

[Klassner 93]
F. Klassner, V. Lesser et H. Nawab. *Fusing multiple reprocessings of signal data.* The Proceedings of the SPIE Sensor Fusion VI Conference, vol. 2059, Boston, Massachusetts, Septembre 1993.

[Laasri 89]
H. Laasri et B. Maitre. *Coopération dans un univers multi-agents basée sur le modèle du blackboard : études et réalisations.* Thèse de doctorat de l'Universoté Henri Poincaré, Nancy I, 1989.

[Labidi 93]
S. Labidi et W. Lejouad. *De l'Intelligence Artificielle distribuée aux systèmes multi-agents.* Projet SECOIA, Rapport de recherche n°2004, 1993.

[Lallement 96]
Y. Lallement. *Intégration neuro-symbolique et intelligence artificielle. Applications et implantation parallèle.* Thèse de doctorat de l'Université Henri Poincaré, Nancy I, 1996.

[Lambolais 98]
T. Lambolais, N. Lévy et J. Souquières. *Assistance au développement de spécifications de protocoles de communication.* Technique et Science Informatiques, vol. 17 (9), Novembre 1998.

[Lander 92]
S. E. Lander et V.R. Lesser. *Customizing distributed search amoung agents with heterogeneous knolledge.* Computer Science Department, University of Massachusetts, Amherst, 1992.

[Lee 95]
E. A. Lee et D. G. Messerschmitt. *System-level Design Methodology for Embedded Signal Processors.* The 1995 Annual RASSP Report, University of

California, Berkeley, Septembre 1995.

[Lee 98]
E. A. Lee. *Overview of The Ptolemy Project.* ERL Technical Report UCB/ERL n° M98/71, University of California, Berkeley, CA 94720, Novembre 1998.

[Lesser 77]
V.R. Lesser et L. Erman. *A retrospective view on the HEARSAY-II architecture.* Blackboard Systems, R. Engelmore et T. Morgan, 1988.

[Lesser 88]
V.R. Lesser, J. Pavlin et E.H. Durfee. *Approximate processing in real-time problem solving.* Artificial Intelligence, 9(1). pp 49-61, 1988.

[Lesser 93]
V.R. Lesser, H. Nawab, I. Gallastegi et F. Klassner. *IPUS: An architecture for integrated signal processing and signal interpretation in complex environnements.* The Proceedings of the 1993 National Conference on Artificial Intelligence (AAAI-93), pp 249-255, Washington, DC, Juillet 1993.

[Lesser 94]
V.R. Lesser, H. Nawab et F. Klassner. *IPUS: An architecture for the integrated processing and understanding of signals.* TR 93-29, Computer Science Department, University of Massachusetts, Amherst, Mars 1994.

[Lévy 94]
N. Lévy, A. Dekdouk, N. El Cadi, J.-M. Hufflen, P. Kaboré, T. Lambolais, A. Schaff et J. Souquières. *Aide à la construction et à la réutilisation de spécifications formelles.* Contrat CNET-CNRS, Rapport Technique n°1, Avril 1994.

[Lieberman 95]
H. Lieberman. *Letizia : An Agent That Assists Web Browsing.* IEEE Symposium on Visual Languages, Darmstadt, Germany, Septembre 1995.

[Lieberman 96]
H. Lieberman et D. Maulsby. *Instructible agents : Software that just keeps getting better.* IBM Systems Journal, vol. 35, n°3 & 4, 1996.

[Lieberman 97]
H. Lieberman. *Autonomous Interface Agents.* ACM Conference on Human-Computer Interface [CHI-97], Atlanta, Mars 1997.

[Lieberman 98]
H. Lieberman. *Integrating User Interface Agents with Conventional Applications.* Proceedings of the ACM Conference on Intelligent User Interfaces, San Francisco, Janvier 1998.

[Madisetti 95]
V. Madisetti et J. Corley. *Rapid Prototyping of Application Specific Signal Processors : Current Practices, Challenges, and Roadmap.* The RASSP Digest, vol. 2, 2nd Quarter 1995.

[Marcos 98]
M. Marcos, S. Moisan et A. P. del Pobil. *Knowledge Modeling of Program Supervision Task.* Proceedings of the 11th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems (IEA-98-AIE), Proceedings Springer Verlag – Lecture Notes in Computer Science 1415, vol. 1, pp. 124-133, Benicassim, Espagne, Juin 1998.

[Mari 96]
J-F Mari et A. Napoli. *Aspect de la classification.* Rapport interne, CRIN 96-R-072, 1996.

S.C.D. - U.H.P.
NANCY 1
SCIENCES
BIBLIOTHÈQUE

[Mensch 93]
A. Mensch, D. Kersual, A. Crespo et F. Charpillet. *REAKT Architecture.* The Proceedings of the workshop on integration in real-time intelligent control systems, Madrid, Espagne, Octobre 1993.

[Mensch 95]
A. Mensch et F. Charpillet. *The REAKT Project : environment and methodology for the development of real-time knowledge-based systems.* The Proceedings of the INRIA/IEEE Conference on emerging technologies and factory automation, Paris, Octobre 1995.

[Mensch 96]
A. Mensch et F. Charpillet. *Scheduling in the REAKT kernel : combining predictable and undounded computations for maximizing solution quality in real-time knowledge based systems.* The Proceedings of the RTS&ES'96, Paris, Janvier 1996.

[Moisan 97]
S. Moisan, R. Vincent et M. Thonnat. *Program supervision : from knowledge modeling to dedicated engines.* Projet ORION, Rapport de recherche n°3324, Décembre 1997.

[Moisan 98]
S. Moisan. *Une plate-forme pour une programmation par composants de systèmes à base de connaissances.* Habilitation à diriger les recherches, Université de Nice, 1998.

[Moisan 00]
S. Moisan et D. Ziébelin. *Résolution de problèmes en pilotage de programmes.* RFIA 2000, 12ème Congrès Reconnaissance des Formes et Intelligence Artificielle, Paris, France, Février 2000.

[Mouaddib 93]
A-I. Mouaddib. *Contribution au raisonnement progressif et temps-réel dans un univers multi-agents.* Thèse de doctorat de l'Université Henri Poincaré, Nancy I, Octobre 1993.

[Mouaddib 98]
A-I. Mouaddib et S. Zilberstein. *Optimal Scheduling of Dynamic Progressive Processing.* Proceedings of the 13th Biennial European Conference on Artificial Intelligence (ECAI-98), Brighton, Royaume-Uni, 1998.

[Musen 89]
M.A. Musen. *Automated Generation of Model-Based Knowledge-Acquisition Tools.* Morgan Kaufmann, Inc., San Mateo, Calif., 1989.

[Musen 93]
M.A. Musen et S.W. Tu. *Problem-Solving Models for Generation of Task-Specific Knowledge-Acquisition Tools.* Knowledge Oriented Software Design, eds., J. Cuena, North-Holland (Amsterdam), pp. 23-49, 1993.

[Nawab 94]
H. Nawab et V.R. Lesser. *Integrated processing and understanding of signals.* Symbolic and knowledge-based signal processing, Prentice Hall signal processing series, chapitre 7, 1994.

[Nii 82]
H. Nii, E. Feigenbaum, J. Anton et A. Rockmore. *Signal-to-symbol transformation : HASP/SIAP case study.* AI Magazine, vol. 3, 1982.

[Oppenheim 94]
A. V. Oppenheim et S. H. Nawab. *Symbolic and knowledge-based signal*

processing. Prentice Hall signal processing series, préface, 1994.

[Parmentier 98]
T. Parmentier, D. Ziebelin, F. Rechenmann. *Environnement de résolution de problèmes distribué.* Actes du 11ème Congrès Reconnaissance de Formes et Intelligence Artificielle (RFIA'98), Clermont-Ferrand, France, janvier 1998.

[Parmentier 99]
T. Parmentier. *Distributed Problem Solving Environment Dedicated to the DNA Sequence Annotation.* Proceedings of the 11th European Workshop on Knowledge Acquisition, Modeling, and Management (EKAW'99), Dagstuhl Castle, Germany, May 1999.

[Pierret-Golbreich 91]
C. Pierret-Golbreich. *Vers une Nouvelle Architecture pour l'Explication du Raisonnement au Niveau des Connaissances :TASK.* Rapport technique, RR674, Université Paris-Sud (Orsay)/LRI, juin 1991.

[Quinlan 88]
J. R. Quinlan. *Decision trees and multi-valued attributes.* Machine Intelligence, Oxford University Press, chapitre 11, pp. 305-318, 1988.

[Quinlan 90]
J. R. Quinlan. *Decision trees and decision making.* IEEE transactions on systems, man and cybernetics, vol. 20, n°2, 1990.

[Quinlan 93]
J. R. Quinlan. *C4.5 programs for machine learning.* Morgan Kaufmann publishers, 1993.

[Rice 96]
J. R. Rice et R. F. Boisvert. *From Scientific Software Libraries to Problem-Solving Environments.* Proceedings of IEEE Computational Science & Engineering, Purdue University, pp. 44-52, September 25-27, 1996.

[Rousseau 88]
B. Rousseau. *Vers un Environnement de Résolution de Problèmes en Biométrie : Apports des Techniques de l'Intelligence Artificielle et de l'Interaction Graphique.* Thèse de doctorat de l'Université Claude Bernard, Lyon I, Février 1988.

[Sebillotte 87]
S. Sebillotte. *La planification hiérarchique comme méthode d'analyse de la tâche : analyse de tâches de bureau.* Rapport de recherche n°599, INRIA, Février 1987.

[Sichman 94]
J. S. Sichman, Y. Demazeau, R. Conte et C. Castelfranchi. *Un mécanisme de raisonnement social fondé sur des réseaux de Dépendance.* 2ème Journées Francophones IAD&SMA, Voiron, Mai 1994.

[Schreiber 94]
G. Schreiber, B. J. Wielinga et R. de Hoog. *CommonKADS: A comprehensive methodology for KBS development.* IEEE Expert, vol. 9, n°6, pp. 28-37, 1994.

[Shekhar 94]
C. Shekhar, S. Moisan et M. Thonnat. *Use of a Real-Time Perception Program Supercisor in a Driving Scenarion.* Intelligent Vehicle '94, Paris, 24-26 Octobre 1994.

[Steels 90]
L. Steels. *Components of Expertise.* Artificial Intelligence, vol. 11, 2, pp 28-49, 1990.

[Tira 96]
Z. Tira. *Apport de l'analyse multirésolution oblique et du rapport de vraissemblance généralisé à la reconnaissance partiellement coopérative.* Rapport d'activité EDF, 1996.

[Thonnat 93]
M. Thonnat, S. Moisan et J. Van Den Elst. *Supervision of Perception Tasks for Autonomous Systems : the OCAPI Approach.* Programme 3 & 4, Rapport de recherche n°2000, Juin 1993.

[Thonnat 98]
M. Thonnat et S. Moisan. *What can Program Supervision Do for Software Re-use ?* IEA-AIE'98, 11th International Conference on Industrial & Engineering Applications of Artificial Intelligence & Expert Systems, Benicasim, Espagne, Lecture Notes in Artificial Intelligence 1415, June 1998.

[Willamowski 92a]
J. Willamowski et F. Jean-Marie. *Modélisation de dialogues et de tâches pour le développement de systèmes coopératifs en calcul scientifique.* Projet SCARP, Manuel utilisateur, Novembre 1992.

[Willamowski 92b]
J. Willamowski et F. Jean-Marie. *Modélisation de dialogues et de tâches pour le développement de systèmes coopératifs en calcul scientifique : Représentation des connaissances pour l'analyse modale.* Projet SCARP, Décembre 1992.

[Willamowski 94]
J. Willamowski. *Modélisation de tâches pour la résolution de problèmes en coopération système-utilisateur.* Thèse de doctorat de l'Université Joseph Fourier, Grenoble I, 1994.

[Wielinga 92]
B. J. Wielinga, A. Th. Schreiber et J. A. Breuker. *KADS: a modelling approach to knowledge engineering.* Knowledge Acquisition, n°4, pp. 5-53, 1992.

[Williams 90]
M. Williams. *Hierarchical multi-expert signal understanding.* Blackboard systems, R. Engelmore et A. Morgan, éditions Addison-Wesley, 1990.

[Winograd 96]
J. M. Winograd, J. T. Ludwig, S. H. Nawab, A. V. Oppenheim et A. P. Chandrakasan. *Flexible Systems for Digital Signal Processing.* AAAI Fall Symposium on Flexible Computation, Cambridge, MA, Novembre 1996.

Monsieur **HAIK Philippe**

DOCTORAT de l'UNIVERSITE HENRI POINCARE, NANCY-I

en INFORMATIQUE

VU, APPROUVÉ ET PERMIS D'IMPRIMER

Nancy, le 18 JAN. 2000 n° 341

Le Président de l'Université

UNIVERSITÉ HENRI POINCARÉ NANCY
LE PRESIDENT

Cl. BURLET

C-F9

UNIVERSITÉ HENRI POINCARÉ - NANCY 1
24-30, rue Lionnois B.P. 3069 54013 Nancy cedex - Tél. 83 85 48 00 - Fax 83 85 48 48

Adresse électronique : _____ @presn1.u-nancy.fr