Résolution de problèmes en pilotage de programmes.

Problem Solving in Program Supervision.

S. Moisan[1] D. Ziébelin[2]

[1] INRIA Sophia Antipolis BP 93
[2] INRIA Rhône Alpes

06902 Sophia Antipolis Cedex
Sabine.Moisan@sophia.inria.fr

Résumé

Les techniques de pilotage de programmes permettent de capitaliser les connaissances sur l'utilisation de programmes complexes et d'opérationnaliser cette utilisation pour des utilisateurs non spécialistes en leur apportant une aide pour le choix, le paramétrage et l'enchainement des programmes. Cet article analyse différents systèmes de pilotage de programmes, dégage un modèle commun et identifie les types de connaissances nécessaires. Nous détaillons comment cette connaissances est modélisée, représentée et prise en compte dans nos systèmes, plus particulièrement dédiés au pilotage de programmes scientifiques et techniques. Enfin, nous envisageons les évolutions des principes du pilotage afin de prendre en compte, au moment de l'exécution, des contraintes de ressources.

Mots Clefs

pilotage de programmes, systèmes à base de connaissances, limitation de ressources, mémoire technique.

Abstract

Techniques of program supervision favour the capitalization of knowledge about the use of complex programs and the operationalization of this utilization for users non specialist in the domain offering them help concerning the choice, parametrizing and sequencing of programs. This paper analyses different program supervision systems, brings out a joint model, and identifies the necessary knowledge for program supervision. We detail how this knowledge is modeled, represented and used in our systems, mostly for scientific computing program supervision. Finally, we study the evolution of program supervision principles in order to take into account resource constraints during the execution.

Keywords

program supervision, knowledge-based systems, resource limitations, technical memory.

1 Introduction

Le pilotage de programmes vise à l'utilisation optimale d'un ensemble de programmes préexistants, indépendamment d'un domaine applicatif particulier. Pour cela, les systèmes de pilotage automatisent – partiellement ou complètement – les étapes nécessaires à la construction d'une chaîne de programmes de traitement qui résout l'objectif d'un utilisateur. Le but du pilotage n'est pas d'optimiser les programmes eux-même, mais leur *utilisation*, qui peut être délicate pour un utilisateur novice. En effet, les programmes ne s'appliquent que sous certaines conditions, qui dépendent par exemple de la nature ou du contenu de leurs données d'entrée. Leur syntaxe d'appel peut être complexe et leur exécution nécessite souvent la détermination préalable de valeurs de paramètres d'entrée ou le formatage de certaines données.

La motivation des systèmes de pilotage est double : gestion de l'utilisation opérationnelle de programmes et capitalisation des compétences pour utiliser optimalement ces programmes. Les techniques à base de connaissances, adoptées pour réaliser les systèmes de pilotage permettent de concilier ces aspects *résolution opérationnelle* de problèmes de réutilisation de logiciels et *capitalisation* de connaissances sur cette (ré)utilisation.

On distingue deux types d'intervenants dans un système de pilotage de programmes :

- un ou des experts sur les programmes, qui construisent une base de connaissances sur l'utilisation et les caractéristiques des programmes

– un ou des utilisateurs « applicatifs », qui utilisent le système pour une application donnée, en spécifiant un problème particulier et un objectif de traitement. L'utilisateur peut être un autre logiciel, comme dans le cas d'un système embarqué.

La démarche de construction d'un système de pilotage consiste à opérationnaliser les méthodes de résolution de problèmes connues de l'expert et à rendre exécutable sa connaissance sur l'utilisation des programmes. Plusieurs approches peuvent être utilisées pour expliciter ces méthodes de résolution. Certaines approches permettent par exemple d'identifier des méthodes de résolution de problèmes (*Problem Solving Methods*), comme les « Tâches Génériques » [3] ou le système PROTÉGÉ [19]. De même, les approches comme KADS [24] contribuent à la construction de modèles qui seront ensuite opérationalisés au travers d'un système de pilotage.

La gestion automatique de l'utilisation de programmes est rendue nécessaire par l'existence de plus en plus de (bibliothèques de) programmes, difficiles à utiliser sans connaissances dans le domaine traité par les programmes, et qui servent souvent dans des applications d'un domaine différent. C'est le cas par exemple de bibliothèques de traitement d'images très utilisées dans des domaines tels que la médecine, la biologie, l'astronomie, etc., ou encore dans des chaînes de traitement automatiques. Les systèmes de pilotage déchargent, partiellement ou totalement, l'utilisateur de la gestion concrète des programmes. Ils gèrent pour lui la sélection des meilleurs programmes, leur ordonnancement, la gestion des flots de données entre programmes, la détection de problèmes et les retours en arrière nécessaires, en tenant compte des particularités des algorithmes, de leurs implantations, des formatages intermédiaires des données, des syntaxes d'appel complexes avec divers paramètres à régler, etc. Un système de pilotage offre ainsi des capacités de reconfiguration et d'adaptation qui sont particulièrement cruciales lors d'utilisations dans des chaînes de traitement automatiques. Dans certains cas, un dialogue avec l'utilisateur applicatif peut s'établir sur des points relevant de sa compétence, afin de guider le processus de pilotage (par exemple, pour évaluer la qualité d'un résultat de traitement).

Les techniques de pilotage permettent aussi de constituer une mémoire consultable des compétences sur l'utilisation de programmes existants. Cela répond à un besoin des entreprises qui, en cas d'indisponibilité ou de départ des experts possèdant ce savoir faire, souhaitent de plus en plus conserver les connaissances sur les traitements, sous une forme compréhensible et si possible opérationnelle. Ces informations, rarement explicitées, ne se trouvent ni dans les algorithmes ou les codes sources. L'explicitation de cette connaissance facilite les reconceptions et les modifications.

Il existe différentes dénominations, correspondant à des formes plus ou moins sophistiquées de pilotage, telles que « configuration » [1, 15], « planification de tâches » [14], « composition de processus » [12], « construction de scripts exécutables » [5], « aide à l'application d'une bibliothèque de codes » [13]. Même si les travaux sont issus d'un domaine particulier, comme le traitement d'images, ils aboutissent souvent à des systèmes généraux. En effet, quel que soit le domaine, la sélection et l'assemblage des programmes les mieux adaptés à un problème et leur application correcte correspond à la même forme de résolution de problèmes, bien qu'il existe des variantes sur certains points.

La problématique du pilotage de programmes est très générale et a été appliquée dans des domaines divers, comme le traitement d'images, le traitement du signal, l'automatique, etc. (voir un état de l'art dans [21]). Il existe ainsi de plus en plus de systèmes de pilotage dans differents domaines. Cependant, aucun n'a abordé le problème de la limitation de ressources. Pourtant, l'utilisation de systèmes de pilotage dans des contextes industriels implique une gestion stricte de divers types de ressources, comme le temps, les ressources matérielles et logicielles, l'espace mémoire ou les données partagées. Ces ressources constituent alors une nouvelle catégorie de connaissance qui doit être prise en compte durant le processus de pilotage.

L'objectif de cet article est tout d'abord (section 2) de dégager un modèle commun issu de différentes approches et d'identifier les types de connaissances expertes nécessaires. Nous nous basons pour cela sur des systèmes de pilotage, comme VSDE [1], MVP [5], VISIPLAN [12], BORG [7], SOLUTION [15], SPICE [13], EXTI [10], MMIIS [11], ou sur des travaux proches, comme COLLAGE/KHOROS [14], ainsi que sur nos travaux [6, 22, 20], qui portent plus particulièrement sur le pilotage de programmes scientifiques. La section 3 examine ensuite en détails les deux types de systèmes de pilotage sur lesquels nous avons travaillé et reprend les principales phases de l'utilisation des connaissances expertes par le mécanisme de résolution d'un moteur de pilotage. Enfin, la section 4 examine en quoi la prise en compte des limitations de ressources peut s'intégrer dans nos modèles et les faire évoluer.

2 Description des environnements de pilotage

Les systèmes de pilotage sont, comme tous les systèmes à base de connaissances, composés d'une base de connaissances et d'un moteur de résolution (voir figure 1). Une base de connaissances en pilotage contient principalement les représentations rp_i et rc_j des programmes p_i et de leurs configurations possibles c_j, connues des experts (en totalité ou en partie). Le contenu de ces représentations doit être suffisant pour permettre au moteur de sélectionner les programmes, d'initialiser les valeurs de leurs paramètres, de gérer les flots de données entre programmes et de produire un enchaînement de programmes adapté aux données de l'utilisateur et à ses contraintes.

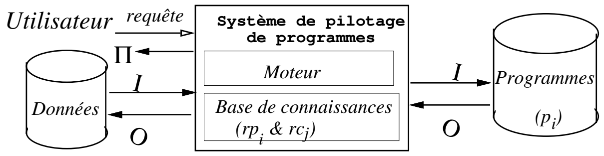

FIG. 1 – *Système de pilotage : l'utilisateur fournit une requête (composée principalement d'un objectif de traitement de haut niveau et de contraintes sur les résultas souhaités) et les données d'entrée $\mathcal{I}$. Le système de pilotage construit l'enchaînement Π de programmes qui produit les données de sortie $\mathcal{O}$. Pour cela, il utilise les informations de la base de connaissances, principalement les représentations rp_i et rc_j.*

Les représentations rp_i sont attachées aux programmes (ou modules, ou fonctions) correspondant à des codes existants. Elles contiennent des informations permettant l'exécution des programmes associés. Elles sont souvent connectées à des représentations des arguments en entrée et en sortie des programmes, décrivant chaque argument par son type, son domaine de validité ou son format. Les rc_j représentent des configurations prédéfinies, connues des experts du domaine, qui peuvent se décomposer de diverses façons, en sous-niveaux de plus en plus concrets. Elles contiennent des informations permettant de raffiner un enchaînement pour passer d'un haut niveau d'abstraction vers des niveaux inférieurs. Nous nommerons ces deux types de représentations sous le terme de *tâches* (respectivement primitives et composites) dans la suite.

De plus, le pilotage de programmes implique des points de décisions dans la résolution, qui peuvent être renseignés par l'expert dans la base de connaissances, via des critères de décision indiquant comment sélectionner une tâche, comment décider d'un échec ou d'un problème, comment régler les valeurs des paramètres des programmes, comment choisir entre des alternatives, etc. Ces critères sont utilisés par le moteur pour diriger son raisonnement. Ce sont eux qui confèrent aux systèmes leur flexibilité et leur robustesse face à des changements de situations.

Plus formellement, étant donnés :
- $\mathcal{P}$ ={ p_i/i $\in$ 1..n } un ensemble de programmes p_i, (codes exécutables existants) ;
- {$rp_i \cup rc_j$} un ensemble de représentations rp_i des programmes p_i, plus un ensemble (éventuellement vide) de représentations rc_j de configurations utiles c_j à partir des programmes de base;
- {cr_k} un ensemble de critères décisionnels ;
- $\mathcal{I}$ un ensemble de données d'entrée (il s'agit de données réelles, fournies par l'utilisateur final pour un problème particulier) ;
- $\mathcal{EO}$ un ensemble de données de sortie attendues (dont on connait seulement le type et le nombre) ;
- $\mathcal{C}(\mathcal{EO})$ un ensemble de contraintes sur les données de sortie attendues ;

un système de pilotage fournit en sortie :
- Π ={p_k /k $\in$ 1..m, m $\leq$ n, $p_k \in \mathcal{P}$ et il existe un ordre partiel sur les p_k }, un enchaînement de programmes (où il peut y avoir plusieurs occurences du même p_k peut apparaître plusieurs fois), avec les flots de données établis entre programmes ;
- et $\mathcal{O}$ un ensemble de données de sortie réelles, tel que : $\mathcal{O}= \Pi(\mathcal{I})$ et $\mathcal{C}(\mathcal{O})$ sont vérifiées.

2.1 Connaissances pour le pilotage

Les connaissances intervenant dans le raisonnement en pilotage de programmes sont variées, et peuvent se représenter de diverses façons. En relation avec le

Primitives rp_i		
Fonctionalité	seuillage d'images	alignement de sé- -quences génomiques
Entrées		
Données	image1 à seuiller	. séquence génomique . banque de criblage
Paramètres	seuil	. pourcentage d'iden- -tification requis . coefficient de mise en correspondance
Sorties	image2 seuillée	fichier résultat
Préconditions	bruit image1 =gaussien	
Appel	cd *image1*.path; muls -s *seuil image2*	blast *banque sequen- -ce pourcentage*...

FIG. 2 – *Vue abstraite de tâches primitives dans deux domaines : traitement d'images et analyse génomique. Une tâche primitive réalise une fonctionalité (e.g., un seuillage). Elle a des arguments en entrée et en sortie. Son exécution dépend de préconditions, par exemple sur le format de ses entrées. Des critères associés (voir figure 4) permettent de régler la valeur des paramètres ou d'évaluer la qualité des résultats. Sa syntaxe d'appel sera instanciée avec les valeurs des arguemnts (en italiques) à l'exécution.*

modèle formel défini ci-dessus, nous allons détailler les principaux concepts et en donner des illustrations.
Tout d'abord, les descriptions des programmes, des configurations prédéfinies (rp_i et rc_j), ainsi que des données et des arguments des programmes sont souvent réalisées par des frames, qui stockent la connaissance correspondante de façon lisible et utilisable par le mécanisme de résolution de problèmes. Une vue abstraite du contenu de la description d'une tâche primitive est déclinée dans deux domaines différents (l'analyse génomique et le traitement d'images) figure 2.
Selon les systèmes, les opérateurs de configuration possibles pour les rc_j sont plus ou moins riches (voir figure 3). Ainsi, on a toujours des opérateurs de séquences, de choix ou de spécialisation en sous-tâches, mais plus rarement des opérateurs d'itérations ou d'optionnalité.

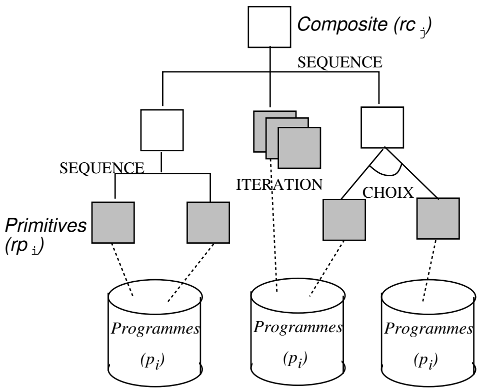

FIG. 3 – *Une tâche composite et ses sous-tâches. Les tâches composites sont représentées par des carrés blancs (rc_j), les primitives (rp_i) en grisé. Il s'agit d'une configuration en séquence au premier niveau, qui enchaîne une séquence, une itération et un choix au deuxième niveau. Les tâches primitives encapsulent des programmes p_i, qui peuvent appartenir à différentes bibliothèques.*

Enfin, les différents critères cr_k de sélection, de choix, etc. sont implantés soit par des règles, dans des bases de règles [6], soit par des dialogues prédéfinis avec l'utilisateur [1], soit à l'aide de mesures sur les données [1, 15]. Les règles peuvent prendre diverses formes : bases de règles attachées aux frames représentant le programme [6] (voir des exemples figure 4), ou règles commençant par une prémisse discriminante [13].
En résumé, les tâches primitives ou composites et les critères sont les principaux constituants de toute base de connaissances en pilotage de programmes. Ils contiennent l'information minimum nécessaire et suffisante pour le pilotage. Mais quelques informations supplémentaires - sur le domaine d'application, par exemple - peuvent apporter un plus qui va faciliter le raisonnement. Ces connaissances interviennent dans les différentes phases de la résolution de problème.

Regle PEGASE (exemple en analyse genomique)
Initialization criteria
Rule name: init-coeff
  comment: "Si le pourcentage requis est petit,
    calcul du coef. de mise en correspondance"
  If pourcentage < 5
  Then correspondance := ((pourcentage-5)
    * pourcentage)/100
Regle PEGASE (exemple en traitement d'images)
Evaluation criteria
Rule name: r-3-eval
  comment: "Si la taille de l'objet est trop petite
    la detection est ambigue"
  Let *context* **a** Context
  If size < context.size-max
  Then assess-operator ambiguous **repair**

FIG. 4 – *Exemples de critères dans la syntaxe* PEGASE *(les mots clef du langage d'expression des connaissances sont en gras). En haut, une règle d'initialisation, correspondant à l'exemple précédent en analyse génomique. En bas, une règle d'évaluation correspondant à l'exemple précédent en traitement d'images. Ces règles font partie des critères attachés aux tâches.*

2.2 Modèle général du mécanisme de résolution

Le mécanisme de résolution adopté par le moteur vise à émuler le comportement d'un expert lors de l'utilisation des programmes. Selon les systèmes, les techniques d'intelligence artificielle utilisées diffèrent. Elles sont principalement à base de différentes formes de planification comme dans VISIPLAN et COLLAGE, où l'enchaînement de programmes à trouver est vu comme un plan, mais on trouve aussi des techniques de classification, comme dans SCARP [25] ou d'instanciation de hiérarchies de tâches, comme dans VSDE [1].
Dans un modèle général du pilotage, ce mécanisme se décompose en différentes phases :

1. une phase préliminaire optionnelle d'*identification* de la tâche de traitement à réaliser, qui à partir de la requête utilisateur va déterminer principalement le but de traitement en termes de pilotage,
2. une phase de *construction* d'une proposition de solution, qui sélectionne et assemble des programmes, pour construire un(e partie d') enchaînement exécutable, voire pour générer réellement un code ; les deux types de tâches, composites ou primitives, peuvent coexister à un moment donné dans une proposition de solution, mais seules les tâches

primitives sont acceptées dans une solution.

3. une phase d'*exécution* effective de l'enchaînement des codes eux-mêmes, après réglage des valeurs de leurs paramètres, qui produit des résultats,
4. une phase d'*évaluation* des résultats, qui détecte d'éventuels problèmes, c'est-à-dire, selon les systèmes, une erreur d'exécution, une information manquante, une donnée incomplète ou un résultat de mauvaise qualité ; cette phase concerne principalement les données de sorties, elle peut être interactive ou automatique ; si aucun problème n'est détecté, le processus continue,
5. en cas de mauvaise évaluation, une phase de *réparation* décide des actions de correction appropriées à partir des problèmes détectés et de la proposition de solution courante ; la réparation peut revenir sur des choix, changer l'ordre des programmes ou changer les valeurs de paramètres de certains programmes ; la réparation peut être interactive ou automatique.

Ce processus s'arrête lorsque une proposition est déclarée solution, par l'utilisateur ou le système (grâce à un critère expert), ou lorsque le système est bloqué et ne peut plus faire de nouvelle proposition.

Les connaissances décisionnelles fournies par l'expert dans les critères, de même que les connaissances contenues dans les descriptions des programmes et des données, jouent des rôles différents à différents moments de la résolution, en interaction avec les mécanismes de résolution fixés par le moteur. Par exemple, la phase d'*exécution* utilise les connaissances sur les programmes de la bibliothèque, leur syntaxe d'appel et leurs paramètres. La phase de *réparation* se sert de critères experts et de stratégies de résolution souvent prédéfinies dans le moteur, qui peuvent consister à interagir avec l'utilisateur.

3 Utilisation des connaissances

La façon d'exprimer la connaissance par l'expert et la façon dont le moteur l'utilise pendant l'exécution sont interdépendantes. Lors de la description des tâches dans une base de connaissances, l'expert doit tenir compte des particularités du mécanisme de résolution du moteur d'inférence qui va les exploiter et des hypothèses stratégiques de celui-ci.

Nous allons examiner les dépendances entre l'expression et l'utilisation des connaissances en nous basant sur nos propres expériences en pilotage. Nous étudions des variantes du modèle du pilotage de programmes qui sont implantées dans divers systèmes : d'une part les systèmes SCARP [25] et PowerTask [20] et d'autre part, le système Ocapi [6] et ses successeurs, basés sur la plate-forme Lama. Ces systèmes partagent de nombreux points communs, notamment sur les choix de représentation des connaissances.

SCARP est un outil générique dans lequel l'accent a été mis sur la coopération avec l'utilisateur, lors de la résolution de problème. Dans PowerTask [20] nous avons étudié les aspects distribués du pilotage, afin d'autoriser l'utilisation de stratégies de résolution et de bibliothèques de programmes sur différents sites.

Le système Ocapi [6] a servi de base pour la partie pilotage de la plate-forme Lama [17], qui fournit un environnement unifié et des composants réutilisables permettant de construire des variantes de systèmes de pilotage. Cette plate-forme facilite l'évolution des systèmes produits. Des moteurs implantant différentes stratégies de pilotage ont été écrits, tels que Pegase [23] ou MedIA [9].

Grâce à ces outils, nous avons réalisé plusieurs systèmes de pilotage, dans des domaines tels que l'astronomie, la stéréo-vision et l'imagerie médicale [22] ou l'analyse génomique [16].

Nous détaillons tout d'abord comment nos systèmes permettent de modéliser les connaissances qui ont été identifiées comme importantes en section 2.1. Puis nous montrerons comment les phases principales du mécanisme de résolution exploitent ces connaissances.

3.1 Représentation des connaissances

Les connaissances sur l'utilisation de programmes s'expriment via les tâches. Nous avons adopté une représentation par objets (frames) pour modéliser les connaissances du domaine de l'application (en particulier les tâches (rp_i, rc_j) et les données qui sont les entrées/sorties des tâches). L'expression des connaissances dans les tâches composites comporte différents aspects. Tout d'abord, des opérateurs de décomposition ou de spécialisation de tâches en sous-tâches permettent de décrire comment raffiner une tâche composite, comme nous l'avons décrit section 2. La modélisation de ce type de connaissances se fait sous forme de hiérarchies de tâches. Cette connaissance est complétée par des informations sur le type des arguments et sur les flots de données. Les flots de données peuvent exister entre les tâches et leurs sous tâches, ou entre tâches du même niveau, dans une hiérarchie de décomposition.

Enfin, des connaissances décisionnelles (cr_k) sont attachées aux tâches à tous les niveaux, pour diriger le mécanisme de résolution et préciser dans quel ordre et comment les tâches seront exécutées. Nous avons choisi de représenter ces connaissances par des bases de règles ou par des conditions attachées aux tâches.

3.2 Mécanisme de résolution

Le mécanisme de résolution exploite les connaissances exprimées dans les tâches, principalement dans les phases d'identification et de construction (section 2.2). Nous reprenons ci-dessous ces deux phases où nos systèmes diffèrent : les systèmes SCARP et POWERTASK utilisent des techniques de classification, alors que les système OCAPI et plus récemment les systèmes de pilotage développés à l'aide de la plate-forme LAMA utilisent des techniques de planification.

La phase d'identification. Dans les systèmes SCARP et POWERTASKS, le mécanisme d'identification est basé sur un algorithme de classification (figure 5) : l'hypothèse stratégique du moteur de pilotage est donc que « la meilleure tâche pour résoudre un problème est celle qui est la plus spécialisée ». Le processus de classification s'effectue en analysant la façon dont les sous-tâches précisent la tâche de plus haut niveau, c'est-à-dire la façon dont les entrées et les sorties d'une tâche mère se spécialisent dans ses sous-tâches. Pour permettre cette classification, le mécanisme d'identification utilise les critères de sélection associés aux tâches composites. Ces critères doivent être exclusifs entre tâches de même niveau. Par ailleurs, du fait qu'ils servent à guider le moteur dans la phase d'identification certains niveaux de classification sont des raffinements liés au domaine d'application. L'un

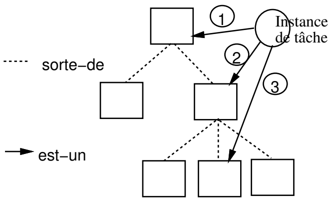

FIG. 5 – *Mécanisme de classification : 1) rattachement de l'instance à sa classe de création. 2) et 3) association de l'instance à la classe la plus spécifique, après vérification des conditions d'appartenance à cette classe, puis possibilité d'inférence sur les attributs non encore déterminés.*

des inconvénients est la prolifération de spécialisations jouant uniquement le rôle du contrôle du moteur de tâches, ce qui est parfois difficile à appréhender pour l'expert du domaine d'application.

La plate-forme LAMA quant à elle, autorise chaque système qu'elle permet de créer à définir sa propre hypothèse stratégique. De manière générale, « la meilleure tâche est celle qui peut être *mise en correspondance* avec la requête de l'utilisateur ». Le mécanisme particulier de mise en correspondance est défini par chaque système et les descriptions fournies par l'expert doivent comporter l'information nécessaire à cette mise en correspondance. Par exemple, dans les systèmes PEGASE et MEDIA, le mécanisme d'identification effectue une mise en correspondance basée sur les fonctionalités de traitement déclarées des tâches et leurs données d'entrées (figure 6). Ces systèmes privilégient les tâches dont la fonctionalité de traitement correspond à l'objectif demandé par l'utilisateur et dont les entrées ont le même nombre, type (ou un super-type) que des données de l'utilisateur. Cela implique que l'expert doit

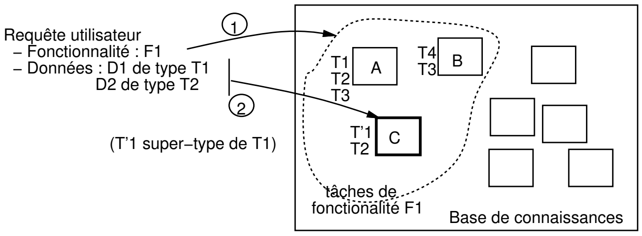

FIG. 6 – PEGASE *se restreint tout d'abord aux tâches ayant la même fonctionnalité que celle demandée par l'utilisateur dans sa requête, puis parmi celles-ci il choisit la (ou les) tâche(s) ayant le même nombre de données d'entrées, avec des types compatibles avec ceux des données de la requête (ici C).*

avoir défini soigneusement les fonctionalités associées aux tâches et les hiérarchies de types des arguments.

La phase de construction. La phase de construction se base généralement sur les rc_j qui permettent, en partant d'un niveau d'abstraction proche de la requête utilisateur, d'arriver jusqu'aux programmes exécutables. Le mécanisme de construction utilise, à chaque étape de raffinement, les critères de choix fournis par l'expert (choix de la meilleure sous-tâche, choix du programme disponible le mieux adapté, etc.), afin de retenir la ou les tâches qui répondent le mieux aux objectifs de l'application.

Dans les systèmes SCARP et POWERTASK, la distinction est faite entre hiérarchies de spécialisation de tâches, et hiérarchies de décomposition de tâches. La phase de construction (voir figure 7) commence par une étape de spécialisation, où les tâches issues des étapes d'identification sont spécialisées (les spécialisations d'une tâche ont une fonctionnalité équivalente, elles peuvent résoudre le même problème, mais de façon différente). Ensuite, vient une étape de décomposition ou d'exécution des tâches sélectionnées par la classification : les tâches composites sont décomposées, jusqu'à une prochaine étape de classification, tandis que pour les tâches primitives, il y a exécution des programmes associés.

Les systèmes PEGASE, OCAPI et MEDIA utilisent une

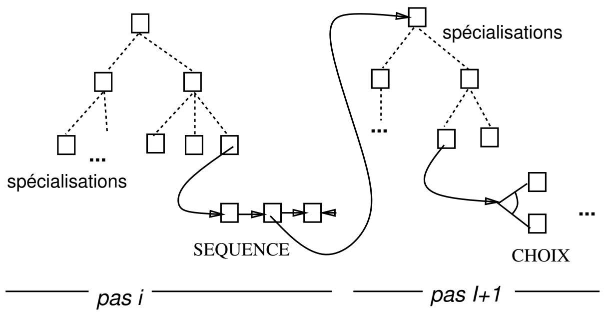

FIG. 7 – *Phase de construction* POWERTASK *: la phase de spécialisation au pas i est suivie d'une phase de décomposition en séquence. Chaque sous-tâche de cette séquence est elle-même soumise à une phase de spécialisation au début de son exécution (au pas i+1).*

seule notion de hiérarchie de tâches, qui inclut spécialisations et décompositions. Dans PEGASE, comme dans OCAPI, la base de connaissances contient une seule hiérarchie de tâches, qui décrit toutes les spécialisations et décompositions possibles, à partir d'une bibliothèque particulière. La planification consiste à trouver le meilleur chemin dans cette hiérarchie, avec des retours en arrière possibles, pour revenir sur certains choix (voir figure 7). Ceci oblige l'expert à avoir

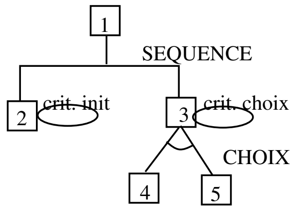

FIG. 8 – *Phase de construction* PEGASE *: la tâche mère 1 est décomposée en la séquence de ses sous-tâches, la tâche 2 (primitive) est exécutée, après application des critères d'initialisation de ses paramètres, le choix de la meilleure sous-tâche de la tâche 3 est fait en utilisant les critères de choix associés, etc.*

une vision exhaustive des configurations des programmes d'une bibliothèque, ce qui peut être fastidieux, ou trop complexe à gérer. C'est pourquoi le système MEDIA ne se base pas sur cette hypothèse contraignante d'une unique hiérarchie de tâches. Lorsque les connaissances de décomposition font défaut, MEDIA passe à une construction à partir de fragments de hiérarchies de tâches plus petites, donc plus faciles à exprimer et à comprendre pour l'expert, ou même à partir des rp_i seules. Les choix se basent alors sur les caractéristiques des données, ou les effets attendus des programmes, qui doivent avoir été renseignés par l'expert, sans quoi le système est bloqué. La base de connaissances peut ainsi évoluer plus facilement, sans devoir repenser toutes les configurations à chaque modification.

4 Intégration de la gestion des ressources

Les programmes peuvent consommer (ou produire) diverses ressources matérielles ou logicielles, comme l'espace mémoire ou les données partagées. La gestion de ces ressources fait intervenir des préoccupations comme l'accessibilité des moyens de calcul et des données (pour la résolution répartie de problèmes), le coût et le temps du transfert de données sur un réseau ou l'existence ou non de dates limites d'exécution. Selon les applications et les programmes, certaines ressources sont plus critiques que d'autres. par exemple, le temps est une ressource fondamentale pour l'exécution de programmes dans un contexte temps-réel, [4] ou lorsque la durée d'exécution des programmes et leurs précédences dans le temps sont critiques pour l'application [26, 18]. Dans de tels contextes, le mécanisme de résolution du pilotage doit s'adapter et prendre en compte ces contraintes de ressources. D'une part, les hypothèses du moteur peuvent changer. Par exemple, le moteur peut imposer de réduire certaines phases coûteuses, comme l'évaluation/réparation, en ne les effectuant qu'à une certaine fréquence, ou seulement lorsque qu'il reste du temps disponible. De plus, de nouvelles connaissances expertes, relatives à la gestion des ressources peuvent aussi être introduites et intervenir au moment des prises de décision du moteur, en plus des autres connaissances de pilotage. Pour permettre l'allocation de ressources aux tâches, les environnements de pilotage doivent posséder en plus des connaissances décrites au paragraphe 2.1, des connaissances sur :

- les ressources nécessaires pour l'exécution des tâches, pour cela, la représentation des tâches de nos systèmes doit être étendue (voir une proposition d'extension, figure 9).
- les ressources disponibles dans l'environnement d'exécution ; nous proposons donc d'ajouter dans les bases de connaissances une représentation des ressources et d'instancier cette représentation au moment d'une résolution,
- les tâches à allouer : ce sont les tâches potentiellement exécutables pour résoudre le problème posé, qui vont être mises en concurrence en fonction des ressources.

En particulier lors des sélections et des choix, le mécanisme de résolution doit privilégier les tâches qui

Primitive
  name : 3D_telemetry
Input Parameters
  Symbol **name :** execution_mode
    range [scanning,tracking]
  ...
Output Data
  Bool **name :** target_lost
  ...
Initialisation Criteria
  if (mode == tracking)
    then mode := scanning *every 5 seconds*
  ...
Hardware: *conflict with Motion_Segmentation*
Duration: *data-dependent, << 200ms*

FIG. 9 – *Exemple d'une description de tâche [18], étendue (parties en italiques) pour la prise en compte de ressources (ici les ressources matérielles et temporelles), dans la représentation de la tâche et dans les critères associés.*

maximisent les objectifs de qualité visés par l'application, tout en minimisant le coût induit par la consommation des ressources. Ce coût se compose du coût d'*allocation* des ressources aux tâches et du coût en temps pris pour calculer cette allocation.

4.1 Allocation de ressources

Le processus d'allocation de ressources doit être intégré à la stratégie de résolution de problèmes utilisée par le moteur de pilotage. Par exemple, dans les systèmes SCARP ou POWERTASKS les ressources nécessaires aux tâches sont décrites et représentées au sein de la base de connaissances au même niveau que les connaissances de domaine. Cette description des ressources peut ainsi être exploitée lors des étapes de classification des tâches effectuée par le moteur de pilotage. Les critères choisis pour effectuer cette classification portent sur les types de ressources nécessaires aux tâches. Dans ce qui suit nous présentons comment s'insère la prise en compte et l'allocation des ressources dans les différentes phases (identification, construction, réparation) du mécanisme de résolution tel qu'il a été présenté au paragraphe 2.2. Nous avons découpé le processus d'allocation de ressources en cinq étapes. Les trois premières étapes s'insèrent après la phase d'identification. Elles utilisent les résultats produits lors de l'identification, c'est à dire la liste des tâches possibles pour résoudre le problème posé. L'étape suivante (4) s'intègre dans la phase de construction, la dernière étape (5) concerne la phase de réparation. Nous détaillons ci-dessous ces étapes.

1. Pour chacune des tâches possibles, détermination les ressources nécessaires à leur réalisation et lier les tâches aux instances d'objets représentant ces ressources.
2. Constitution de la liste des ressources disponibles. Cette étape pourrait être menée en dehors de l'algorithme de résolution, avant la définition du problème. L'effectuer au plus près de la résolution présente l'avantage de pouvoir être beaucoup plus précis quant aux ressources effectivement disponibles et d'éviter que des résolutions concurrentes ne viennent pas s'insérer et consommer des ressources escomptées par les tâches identifiées.
3. Classement des ressources par criticité, à partir du ratio demande/disponiblité.

Ces trois étapes sont générales et s'intègrent dans tout système de pilotage sans différence notable d'un système à l'autre. Ainsi la plate-forme LAMA propose des composants de base pour chacune de ces étapes. Pour le classement des ressources par criticité la stratégie de classement est laissée à définir par chaque système (comme le mécanisme de mise en correspondance, vu au paragraphe 3.2). L'étape suivante s'insère dans la phase de construction, décrite au paragraphe 3.2, elles diffère selon les techniques utilisées.

4. Dans les systèmes SCARP et POWERTASK, le classement des ressources par criticité intervient parmi les critères de spécialisation, lors de la classification des tâches en fonction de la situation courante (voir paragraphe 3.2). Dans notre nouvelle plate-forme, nous proposons d'enchaîner trois classifications sur les trois types de connaissances utilisés (ressources disponibles, ressources nécessaires et tâches à allouer). Ensuite, au moment de l'exécution des tâches primitives, l'allocation réelle des ressources nécessaires a lieu.

   Dans les systèmes PEGASE et MEDIA, à chaque étape de la construction du plan, les choix du planificateur tiennent compte des contraintes de ressources éventuellement indiquées par l'expert via les différents critères et les contenus des rp_i. Les ressources influent sur le processus décisionnel, au même titre que les données. Le classement des ressources par criticité intervient au moment de l'ordonnancement du plan.

Enfin, la dernière étape, qui concerne la phase de réparation, doit s'intégrer dans le mécanisme de réparation propre à chaque système.

5. Retour à l'étape 4 pour raffiner la situation courante, si la tâche choisie n'est pas satisfaisante, en évaluant l'allocation qui a été faite. L'algorithme de construction doit donc proposer une fonctionnalité de retour en arrière qui ne traite pas seulement les conditions d'échec, mais intègre aussi le

critère de criticité des ressources exploité lors de l'étape 4.

En procédant de cette manière l'allocation des ressources est traitée dynamiquement en respectant au mieux le principe central au pilotage de programmes : « décider de façon opportuniste de l'exécution du bon programme, bien configuré, au bon moment ». Cette approche d'allocation de ressources est suffisamment générale pour s'intégrer sans difficultés dans les plateformes de pilotage de programmes que nous avons développées.

La rationalité. Lors des sélections et des choix, le calcul de l'allocation de ressources a un coût et la qualité du choix réalisé peut souffrir du coût nécessaire à cette prise de décision. Il faut donc considérer le coût global de recherche d'une solution en tenant compte du coût de raisonnement et non pas uniquement de la qualité propre de la solution choisie. C'est le problème de la *rationalité*, qui vise à minimiser le coût de la solution (coût d'*allocation* des ressources aux tâches) et de sa recherche (coût en temps pris pour calculer cette allocation). L'introduction de la rationalité dans les systèmes de pilotage de programmes a ainsi pour objectif d'utiliser au mieux les ressources (plus particulièrement le temps de calcul) consacrées au raisonnement [2, 8]. Ainsi I. Crampé, dans [8] préconise, pour SCARP, de construire un modèle qui maximise une fonction dite d'*utilité* dépendant du coût en temps de recherche de la solution et de la qualité de celle-ci. Cette fonction d'utilité associée à une tâche est définie de manière heuristique à partir d'une utilité intrinsèque fixée par l'expert en fonction de son expérience sur l'utilisation de la méthodes des ressources consommées et décroit en fonction du temps nécessaire à la recherche d'une solution. L'algorithme d'évaluation teste les différentes solutions offertes en spécialisation et décomposition dans l'ordre de temps minimum par rapport à une faible consommation de ressources. Dans cette approche, les temps d'exécution nécessaires aux tâches doivent être connus a priori. Pour les tâches décomposables ou spécialisables ils sont estimés selon les règles simples suivantes :

• le temps d'exécution d'une tâche spécialisable est celui de la sous-tâche en laquelle elle est spécialisée,

• le temps d'exécution d'une tâche décomposable en séquence est la somme des temps d'exécution des sous-tâches de sa décomposition

• le temps d'exécution d'une tâche parallèle est le maximum des temps d'exécution de ses sous-tâches,

• le temps d'exécution d'une tâche primitive est identique à celui du code associé.

Ces travaux constituent un premier pas vers la prise en compte de l'allocation de ressources et de la rationalité dans le pilotage de programmes. Nous proposons de poursuivre dans cette direction, en étudiant certains points laissés de côté dans cette première approche, comme la prise en compte des recherches de solutions antérieures et de leur influence sur des recherches futures et une évaluation globale des solutions possibles, plutôt qu'un calcul immédiat pour une tâche à partir des connaissances sur ses sous-tâches directes.

5 Conclusion

Le pilotage permet à la fois la capitalisation des connaissances sur l'utilisation de codes, l'opérationalisation de cette (ré)utilisation par des utilisateurs plus nombreux et l'intégration de cette utilisation dans des systèmes automatiques robustes. Pour cela un système de pilotage utilise divers types de connaissances, en suivant les hypothèses stratégiques, incorporées dans son moteur. Dans cet article nous avons montré les interdépendances entre l'expression de ces connaissances par l'expert et leur utilisation par le mécanisme de résolution du moteur de pilotage.

Dans certaines applications, comme les systèmes répartis ou embarqués, le pilotage peut avoir à tenir compte de diverses ressources limitées, qui doivent s'intégrer dans les mécanismes et les hypothèses des moteurs et dans les connaissances utilisées. Nous avons proposé des enrichissements des modèles de connaissances et des mécanismes de résolution des moteurs, en nous référant aux systèmes sur lesquels nous travaillons. Ces enrichissements sont suffisament généraux pour s'adapter à d'autres systèmes. Nous allons poursuivre l'étude et l'implantation de mécanismes pour la prise en compte de ressources, dans l'objectif de faciliter le travail de modélisation du comportement attendu lors de la résolution, notamment en termes de ressources et de qualité des résultats produits.

Références

[1] R. Bodington. A Software Environment for the Automatic Configuration of Inspections Systems. In *Proceedings of KBUP'95*, pages 99 – 108. INRIA, Sophia Antipolis, INRIA, November 1995.

[2] M. Chaillot. *Une architecture de Contrôle Réactif pour la résolution coopérative de problèmes.* Thèse, Institut Polytechnique de Grenoble, September 1993.

[3] B. Chandrasekaran and T.R. Johnson. Generic Tasks and Task Structures: History, Critique and New Directions. In J. David, J. Krivine, and R.Simons, editors, *Second Generation Expert Systems*, pages 232–272. Springer-Verlag, 1993.

[4] F. Charpillet. A Look on the Issue of Building Real-Time Knowledge-Based Systems. In *AAAI*

Fall-Symposium on Flexible Computation in Intelligent Systems, November 1996.

[5] S. A. Chien and H. B. Mortensen. Automating image processing for scientific data analysis of a large image database. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(8):854–859, August 1996.

[6] V. Clément and M. Thonnat. A Knowledge-based Approach to the Integration of Image Processing Procedures. *Computer Vision, Graphics and Image*, 57(2):166–184, March 1993.

[7] R. Clouard, A. Elmoataz, and M. Revenu. Une modélisation explicite et opérationnelle de la connaissance de traitement d'images. In *RFIA'98, 11ème congrès Reconnaissance des formes et intelligence artificielle*, pages II–65–74, janvier 1998.

[8] I. Crampé. Sémantique des tâches décomposables et spécialisables. Mémoire de DEA d'informatique, INPG-Université Joseph Fourier, Grenoble (FR), juin 1994.

[9] M. Crubézy. *Pilotage de programmes pour le traitement d'images médicales.* Thèse, Université de Nice-Sophia Antipolis, février 1999.

[10] P. Dalle and Ph. Dejean. Planification en traitement d'images : approche basée sur les données. In *RFIA'98, 11ème congrès Reconnaissance des formes et intelligence artificielle*, pages II–75–84, janvier 1998.

[11] A-M. Forte. *Système basé sur la Connaissance pour l'Identification, la Caractérisation et la Mise en correspondance d'entités anatomiques et fonctionelles en Imagerie Médicale Multimodalité.* Thèse, université Francois Rabelais, Tours, 1991.

[12] L. Gong and C. A. Kulikowski. Composition of Image Analysis Processes Through Object-Centered Hierarchical Planning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(10), October 1995.

[13] F. Grimm and H. Bunke. An expert system for the selection and application of image processing subroutines. *Expert Systems*, 10(2):61–71, May 1995.

[14] A. Lansky, S. Friedman, L. Getoor, S. Shmidler, and N. Short. The COLLAGE/KHOROS link: Planning for image processing tasks. In *AAAI Spring Symposium on Integrated Planning Applications*, March 1995.

[15] C.-E. Liedtke, H. Munkel, and U. Rost. Solution for a learning configuration system for image processing. In A.P. del Pobil J.Mira and M. Ali, editors, *11th International Conference on Industrial Applications of Artificial Intelligence and Expert Systems*, Lecture Notes in Artificial Intelligence 1415, Benicassim, Spain, June 1998. Springer-Verlag.

[16] C. Médigue, F. Rechenmann, A. Danchin, and A Viari. Imagene: An Integrated Computer Environment for Sequence Annotation and Analysis. *Bioinformatics*, 2(15), 1999.

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

[18] S. Moisan and M. Thonnat. Program Supervision under Resource Constraints. In *3th European Conference on Artificial Intelligence-ECAI98. Workshop on Monitoring and Control of Real-Time Intelligent Systems.*, August 1998.

[19] M. A. Musen, S. W. Tu, H. Eriksson, J. H. Gennari, and A. R. Puerta. PROTEGE-II: An environment for reusable problem-solving methods and domain ontologies. In *IJCAI*, Chambéry, France, August 1993.

[20] T. Parmentier, D. Ziébelin, and F. Rechenmann. Environnement de résolution de problèmes distribué. In *RFIA'98, 11ème congrès Reconnaissance des formes et intelligence artificielle*, pages II–265–274, janvier 1998.

[21] M. Thonnat and S. Moisan. Knowledge-Based Systems for Program Supervision. In *Proceedings of KBUP'95*, pages 3–8, Sophia Antipolis, France, November 1995. INRIA.

[22] M. Thonnat, S. Moisan, and M. Crubézy. Experience in Integrating Image Processing Programs. In *Internatianal Conference on Computer Vision Systems*, Las Palmas, Canary Islands, January 1999.

[23] R. Vincent. *Etude des mécanismes de gestion des erreurs dans les sytèmes à base de connaissances de pilotage de programmes.* Thèse, université de Nice-Sophia Antipolis, mars 1997.

[24] B. Wielinga, A.Th. Schreiber, and J.A. Breuker. KADS: a Modelling Approach to Knowledge Engineering. *Knowledge Acquisistion*, 4:5–53, 1992.

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

[26] M. Wozniak, P. Taillibert, and P. Vanheeghe. TCLP: a temporal causal link planner with disjunctive temporal relations. In *UK Workshop on Planning and Scheduling*, December 1997.