Un outil d'acquisition et de representation des taches oriente-objet

Christine Pierret-Golbreich, I. Delouis, Dominique L. Scapin

INRIA

UNITÉ DE RECHERCHE
INRIA-ROCQUENCOURT

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P.105
78153 Le Chesnay Cedex
France
Tél.:(1) 39 63 55 11

Rapports de Recherche

N° 1063

Programme 8
Communication Homme-Machine

UN OUTIL D'ACQUISITION ET DE REPRESENTATION DES TACHES ORIENTE-OBJET

Christine PIERRET-GOLBREICH
Isabelle DELOUIS
Dominique L. SCAPIN

Août 1989

RR_1063

Programme 8

UN OUTIL D'ACQUISITION ET DE REPRESENTATION DES TACHES ORIENTE-OBJET

An object-oriented tool for extracting and representing tasks

Christine PIERRET-GOLBREICH, Isabelle DELOUIS, Dominique L. SCAPIN

INRIA Rocquencourt, BP 105
78153 Le Chesnay Cedex , France

pierret @ seti.inria.fr

Résumé

Un concept unificateur de la représentation des connaissances, extension des notions classiques de règle et de méthode est introduit : le concept de *tâche*. L'objectif est de représenter les traitements procéduraux par objets structurés, puis de généraliser la notion d'attachement procédural à des tâches de haut niveau faisant appel à un enchaînement structuré de sous-tâches. Un formalisme de représentation de ce concept, MAD (Méthode Analytique de Description des tâches), permettant la description aussi bien des traitements (tâches de l'utilisateur final) qu'à plus long terme, des modules de raisonnement (tâches de l'expert concepteur) est d'abord proposé. Ce modèle repose sur la définition d'un objet générique *tâche* autorisant à la fois la décomposition "hiérarchique" d'une tâche et la prise en compte des notions de synchronisation et de relations logiques entre tâches. Une première maquette implémentant ce modèle, réalisée en Lelisp, Aïda, Shirka est ensuite présentée. Cet outil est composé de trois modules correspondant à différents niveaux de représentation. Une représentation graphique de la décomposition des tâches permet la description interactive dans le formalisme MAD d'une tâche, quelque soit son niveau de complexité. Une représentation interne à base d'objets permet de modéliser les tâches sous forme d'entités cohérentes et indépendantes. Une représentation intermédiaire assure l'indépendance entre la représentation externe et le modèle objet. Enfin les bénéfices d'une telle approche pour l'acquisition des connaissances sont exposés. Au cours de ce travail plusieurs autres thèmes d'importance majeure aujourd'hui dans le domaine de l'I.A. et particulièrement d'actualité dans le domaine des représentations objets, sont abordés comme ceux de l'évolution de la base de connaissances, des objets composites, de l'acquisition des connaissances.

Mots-clef: - ergonomie - interfaces - conception - formalisme objet - tâche - dynamique des classes - objet composite - acquisition de connaissances -

Abstract

This report introduces a unifying concept for knowledge representation : the concept of task, which is an extension of the classical notions of rules and methods. The aim is to represent procedural processing by structured objects and then generalizing the notion of procedural attachment to high-level tasks which can call a structured concatenation of sub-tasks. Firstly, this report describes a formalism for representing this concept : MAD (Méthode analytique de Description de Tâches i.e; Analytic Method for Task Description) which allows the description both of the final user tasks as well as in the longer term, reasonning modules (expert designer tasks). This model is based on the definition of a generic object task which permits the hierarchical decomposition of a task, with some notions of task synchronisation and logical relationships. A prototype implementing this model, realized in Lelisp, Aïda, and Shirka is then presented. This tool consist of three modules corresponding to different levels of representation. A graphic representation of the tasks decomposition make it possible for a task to be described interactively in the MAD formalism, of any level of complexity. An object-based internal representation allows the tasks to be modeled in the form of independent and consistent entities. An intermediate representation ensures the independance between the external representation and the object model. Finally, the benefits of such an approach for knowledge acquisition are given. Throughout this work, several other themes currently of major importance in the field of Articial Intelligence, and particularly in the domain of object representations are raised such as the evolution of the knowledge base, composite objects and knowledge acquisition.

Keywords : ergonomics, interfaces, design, object formalism, task, class dynamics, composite object, knowledge acquisition

SOMMAIRE

1. Introduction

Ce travail s'intègre dans le cadre d'un projet dont l'objectif général est de développer une approche pluri-disciplinaire pour la conception d'outils intelligents d'aide à la construction d'interfaces homme-machine. Le domaine d'application visé est donc celui de l'assistance à la conception d'interfaces. Dans ce cadre, la définition d'un formalisme de description explicite de la tâche utilisateur et de la représentation que s'en fait l'opérateur est apparue indispensable pour concevoir des interfaces "ergonomiques". En effet du point de vue de l'ergonome, un tel formalisme s'est avéré nécessaire dès lors que l'on souhaitait dépasser les aspects simplement superficiels d'une interface (aspects dits lexicaux et syntaxiques dans les modèles classiques) pour prendre en compte des aspects plus profonds (aspects dits conceptuels et sémantique), comme la tâche-utilisateur à laquelle elle est dédiée. Ce point est détaillé dans (Scapin, 1988), (SCAPIN et al., 1988). Il était donc naturel de s'intéresser en premier lieu au problème de la modélisation des tâches utilisateurs, catégorie essentielle des objets de l'univers pour cette application. Une étude ultérieure sera menée sur le raisonnement de conception d'interfaces. Cette étude devrait conduire à affiner et valider ce formalisme de façon à l'étendre à la modélisation de stratégies de résolution. L'activité de conception d'interfaces, formalisée en termes de tâche et de sous-tâches (modules de raisonnement) servira de support à cette réflexion.

Du point de vue de l' Intelligence Artificielle, la nécessité de structurer la connaissance se fait de plus en plus ressentir. Ainsi est apparue l'utilisation du concept d'objet structuré permettant de disposer de structures de données reflétant les entités conceptuelles de l'univers à modéliser. De nombreux modèles de représentation, dits hybrides, lui ont associé le concept de méthode d'une part et celui de règle d'autre part, offrant ainsi la possibilité de modéliser les connaissances factuelles par les objets, les connaissances procédurales par les méthodes, les connaissances dynamiques et le raisonnement par des règles. Aujourd'hui divers travaux témoignent du besoin de structurer non seulement les connaissances factuelles mais également les connaissances procédurales ou le raisonnement (découpage en sous-blocs logiques de résolution de problème). Un premier indice de cette tendance est fourni par les représentations purement objets qui permettent de décrire sous forme déclarative une méthode puisque, pour ces modèles une méthode est un objet. Un autre indice, est fourni par le constat que pour de nombreux systèmes, il a été nécessaire d'organiser les règles en "paquets", "contextes", ou "tâches" , selon la terminologie respective de leurs auteurs. Il s'avère donc aujourd'hui important, d'introduire un concept structurateur de la connaissance procédurale et du raisonnement, extension d'une part du concept de méthode, d'autre part de celui de règle : le concept de tâche.

Le concept de tâche est tout d'abord décrit, puis la modélisation à base d'objet et son implémentation sont présentés.

2. Le concept de tâche

La nécessité d'introduire le concept de "tâche" est apparue dans différents contextes. On peut distinguer au moins deux optiques d'utilisation de ce concept dont la frontière est parfois floue: la modélisation du raisonnement, et la modélisation d'une activité .

Dans le premier cas, la notion de tâche s'inscrit dans le cadre général de la résolution de problème. Ainsi par exemple dans "The generic task approach" (Chandrasekaran, 1987) il est proposé de construire des systèmes de connaissances à partir de "blocs constitutifs" appropriés à un type spécifique de résolution de problèmes. Chaque tâche générique utilise des formes de connaissances et des stratégies de contrôle qui lui sont propres. Par exemple quatre types de tâches génériques sont distingués pour la résolution de problèmes de classification : "hierarchical classification, abductive assembly, structured matching, database inference". Dans une optique similaire, la catégorie "tâche" est une catégorie prédéfinie de Smeci (1988) qui sert également à modéliser le raisonnement, même si elle ne renvoie pas à des tâches de haut niveau (comme celles de classification ou de conception chez Chandrasekaran).

Dans le second cas, le concept de tâche est utilisé pour modéliser l'activité d'un utilisateur dans un domaine d'application donné (pour plus de détails voir (Scapin et al, 1988b)). Ainsi, les travaux de (Rousseau, 1988), (Pierret, 1988) se situent dans le domaine de l'**assistance aux tâches de calcul scientifique**. Une tâche est formalisée sous forme d'un objet décrivant de manière déclarative les niveaux fonctionnels et opérationnels de la tâche. Par niveau fonctionnel, on entend les conditions d'exécution de la tâche (par exemple type des entrées- sorties) et ses effets. Le niveau opérationnel renvoie à l'action ou aux tâches à exécuter en réponse à un contexte fonctionnel donné. Les travaux de Montalban (1987) Dieng et Trousse (Dieng, 1988) se situent dans le domaine des **tâches de conception**. Le système ABS proposé par M. Montalban fait également intervenir niveau fonctionnel et niveau opérationnel. Dans 3DKAT, le modèle de tâche comporte une dénomination, une classe, des déclarations (liste de types d'objets), des entrées, des sorties, des buts, des préconditions et des actions. Les actions sont constituées d'une liste de procédures, de règles ou de tâches. Les travaux de Michard (1988), Scapin (1988) M. Tueni et J. LI (Tueni et al, 1988) se situent dans le domaine de l'assistance aux **tâches de bureau** bien que leur portée soit plus général : le formalisme MAD proposé par Scapin s'inscrit dans le cadre plus large de la prise en compte de l'ergonomie dans la conception d'interfaces utilisateurs, tandis que le modèle de Michard s'inscrit dans le cadre de la planification et de la représentation des activités de bureau. La première version du formalisme MAD (proposé par Scapin dans (Scapin 1988)), utilise la planification hiérarchique en l'élargissant à la notion de hiérarchie d'items et en y incluant certains aspects de synchronisation. Une tâche est représentée sous forme d'un arbre hiérarchique constitué à partir d'une série d' Items-tâche. Le système AMS proposé par M. Tueni et J. Li fournit un formalisme de représentation des connaissances bureautiques et permet la modélisation de tâches et séquences de tâches à divers niveaux d'abstraction. Les concepts clés sont ceux d'activité et de réseaux d'activités. Une activité représente une tâche de l'utilisateur, elle est caractérisée par des préconditions, des postconditions et une partie exécutive. La partie exécutive d'une activité peut désigner une action (procédure externe agissant directement sur le domaine d'application), un réseau d'activités (séquences de tâches), ou un Mopa (séquence abstraite). Le système génère dynamiquement à partir de chaque séquence abstraite différentes séquences de tâches exécutables adaptées au contexte. Les Mopas permettent de structurer la base de connaissances et autorise le raisonnement à plusieurs niveaux d'abstraction. Les préconditions et les postconditions attachées à une activité sont structurées dans un arbre logique d'états qui explicite une partie de l'état du domaine de l'application. Le formalisme de représentation proposé permet la modélisation des tâches de l'utilisateur mais aussi des tâches systèmes de haut niveau.

On notera que dans la plupart des travaux présentés, la frontière entre modélisation du raisonnement et modélisation d'une activité utilisateur est souvent floue, certaines tâches-utilisateurs s'inscrivant dans le cadre général de la résolution de problème. Il peut donc être intéressant de définir le concept de tâche de manière générique et de distinguer différentes branches issues de cette racine comme le propose par exemple Montalban en distinguant des tâches-système et des tâches-utilisateurs. Des tâches, blocs élémentaires de raisonnement pourraient également être introduites dans cette hiérarchie: classification hiérarchique, agrégation abductive, etc.

3. MAD : un formalisme de description des tâches

Le modèle des tâches proposé dans MAD est dérivé d'une première ébauche de formalisme (Scapin, op. cit., (Scapin et al, 1988)) et inspiré des différents travaux décrits précédemment. MAD propose une extension du concept clé de "hiérarchie" de niveaux d'abstraction, en introduisant la notion de décomposition structurelle suivant une dimension "logico-temporelle". Les principaux concepts introduits dans le formalisme MAD sont ceux de tâche, d'action, et de structure.

3.1 Le modèle de tâche

3.1.1 Définition

La **tâche** est le concept clé du formalisme MAD. Il permet de représenter un traitement quelque soit son niveau de complexité.

Une tâche est définie par les éléments suivants :

- un **état-initial** : sous-ensemble de l'état du monde constitué de la liste des arguments d'entrée de la tâche.
- un **état-final** : sous-ensemble de l'état du monde constitué de la liste des arguments de sortie de la tâche.
- un **but** : sous-ensemble de l'état final, indiquant explicitement le but recherché que l'exécution de la tâche permet d'atteindre.
- des **préconditions** : ensemble de prédicats exprimant des contraintes sur l'état initial qui doivent nécessairement être satisfaites pour déclencher l'exécution de la tâche. On distingue un type particulier de préconditions, les conditions-nécessaires-déclenchantes (C.N.D) qui décrivent des états particuliers qui non seulement doivent être satisfaits pour permettre l'exécution de la tâche mais qui de plus ont un rôle dynamique de déclenchement de la tâche.
- des **postconditions**: ensemble de prédicats exprimant des contraintes sur l'état final qui doivent nécessairement être satisfaites après l'exécution de la tâche.
- un **corps**: niveau opérationnel indiquant *comment* la tâche peut être exécutée.

La syntaxe d'une tâche en MAD est donc:

<tâche>:= <état-initial>, <état-final>, <but>, <préconditions>, <postconditions>, <corps>

La syntaxe des différents éléments, état-initial, état-final etc., est précisée par la suite

3.1.2 Différents niveaux de description d'une tâche

L'explicitation d'une tâche générique d'une part et l'exécution d'une tâche particulière d'autre part ne se situent pas au même niveau d'abstraction. En effet, l'exécution (Par exemple, la réalisation Ouverture-Rapport N°836 de la tâche générique Ouverture-fichier) nécessite l'instanciation de tous les objets sur lesquels la tâche donnée opère (choix du "quoi") ainsi que le choix de la stratégie de résolution (choix du "comment"). Elle fait donc référence à des objets particuliers (par exemple, au fichier "Rapport N°836") et à un cheminement donné dans l'arbre de décomposition de la tâche. En terme d'objets, l'exécution fait appel au niveau instance. A l'opposé, la description d'une tâche générique se situe au niveau classe. La définition de cette classe nécessite deux types d'informations. En effet, cette description fait intervenir à la fois une description générique du type des objets sur lesquels la tâche sera amenée à opérer (niveau schéma ou classe), et des objets particuliers de ces classes, variables universelles appartenant à la classe (niveau instance: voir § 5) permettant de distinguer les objets de même type. Par exemple, la tâche Insertion qui permet d'insérer une région d'un texte donné dans un autre texte, admet deux entrées de type Texte, le texte qui contient la zone à insérer et le texte dans lequel cette zone doit être insérer. Pour décrire cette tâche, l'utilisateur discerne explicitement ces deux textes. Afin de permettre cette distinction et pour assurer la cohérence de la description, les deux textes sont donc distingués nommément dans MAD par la désignation de deux objets différents de type Texte: $texte1 et $texte2. MAD autorise ainsi la présence simultanée de ces deux niveaux d'information (niveau classe et instance) puisque état-initial, état-final, but, ont la syntaxe suivante:

<état-initial> := *état-initial* = , <init> / <état-initial>, <init>

<init> := <nom-objet > *un* <nom-classe>

La description d'une tâche générique fait de plus référence à différentes stratégies possibles d'exécution de le tâche (Fig. 2). MAD permet une telle description puisqu'il autorise la définition d'alternatives.

3.1.3 Exemples :

- La tâche Sélection-région, qui permet de sélectionner une région d'un texte, est définie dans le formalisme MAD, de la façon suivante :

Sélection-région		
état-initial	=	$texte un Texte
état-final	=	$région un Région
but	=	$région un Région
préconditions	=	$texte affiché?
corps	=	séquence : (Marquage-début Marquage-fin)

On note sur cet exemple que les valeurs affectées au champ état-initial (respectivement état-final et but) de la tâche Sélection-région sont constituées du nom de l'objet suivi du type de l'objet. Le nom permet de désigner un objet ($texte, resp. $région) de type donné (Texte, resp. Région). En terme de représentation objet, le nom désigne une instance et le type désigne une classe.

- La tâche Insertion qui permet de copier une région d'un texte donné ($texte1) dans un autre texte ($texte2), est définie par:

Insertion		
etat-init	=	$texte1 un Texte $texte2 un Texte
but	=	$texte2 un Texte
préconditions	=	$texte1 droit-accès-lecture? $texte2 droit-accès- écriture?
corps	=	sequence: (Copie-par-menu Collage-par-menu)

$texte1 et $texte2 ne représentent pas des objets concrets mais des objets génériques pouvant être substitués par des textes quelconques.

3.1.4 Les différents types de tâches

On distingue deux types de tâche selon la nature du corps :

- une **tâche-simple** est une tâche dont le niveau opérationnel est caractérisé par une entité procédurale insécable décrivant l'action à exécuter. Cette notion d'action est représentée par exemple dans Shirka par un objet de type méthode.

- une **tâche composée** est une tâche dont le niveau opérationnel ne fait pas appel à une simple procédure mais à un enchaînement structuré de sous-tâches. Cette combinaison de tâches est décrite par une entité structure permettant de définir à la fois les composants, sous-tâches constitutives de la tâche composite ainsi que leur agencement, c'est-à-dire les relations d'ordre temporel ou logique existant entre ces différents composants.

La syntaxe de corps est donc donnée par:

<corps>:= *corps* = , <action> / <structure>

3.2 Action et Structure

La partie procédurale ou corps est soit une action, procédure élémentaire, soit une structure, combinaison de tâches.

3.2.1 Action

Une action est un traitement simple correspondant à une procédure externe : action physique ou procédure écrite dans un langage informatique quelconque.

3.2.2 Structure

Une structure est caractérisée par la liste des tâches qui la composent, et un constructeur (seq, par, boucle ...) qui décrit l'agencement des tâches impliquées. La syntaxe est donc la suivante:

<structure> := <constructeur>, <taches>
<taches> := <tache> / <tache>, <taches>
<constructeur> := par / seq / alt / boucle

On distingue différents types de structure d'après la nature de leur constructeur: séquence, parallèle, alternative, boucle, conditionnelle. On peut aisément enrichir le formalisme en définissant d'autres types de structure particulières, telle que facultative etc..

- Une **parallèle** représente un ensemble de tâches pouvant être exécutées dans n'importe quel ordre (éventuellement en même temps)
- Une **séquence** représente un ensemble de tâches devant être exécutées l'une après l'autre.
- Les **boucles** représentent des tâches itératives, par exemple tant-que ou avant-que.
- Une **alternative** représente différentes possibilités dans la manière d'exécuter la tâche.

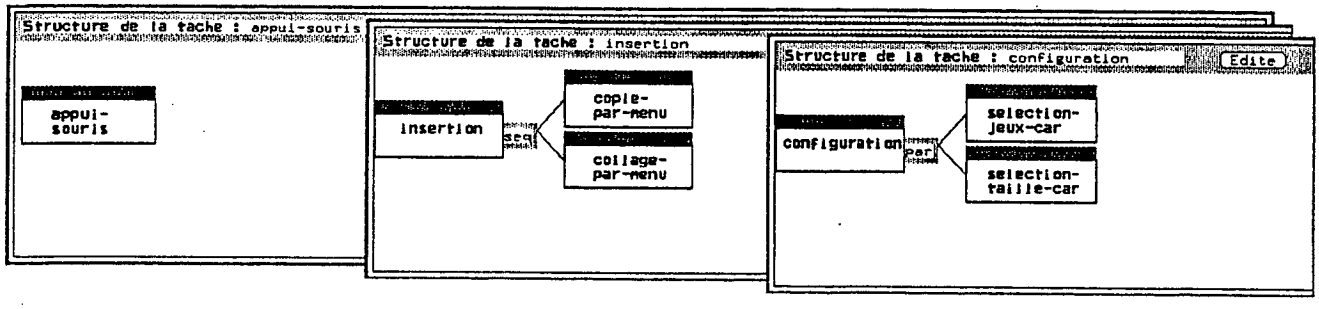


Figure 1 : Exemples de différents types de tâches: simple, séquence, parallèle

3.2.3 Exemples

- La tâche composée Selection-region, qui permet de sélectionner une région dans un texte a pour corps la séquence:

sequence-N°1
constructeur = seq
sous-taches = Marquage-début Marquage-fin

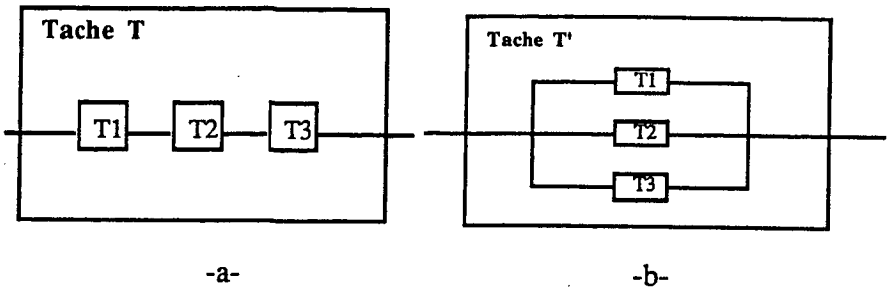
- La tâche simple appui-souris a pour corps l'action appuyer, la tâche simple déplacement-souris a pour corps l'action déplacer[1]

[1] Par convention, les tâches sont désignées par des substantifs et les actions par des verbes

3.3 Représentations graphiques des tâches

Différentes représentations graphiques peuvent être choisies:

- La première correspond à une représentation selon la dimension temporelle (Fig. 2), l'axe représentant le temps.

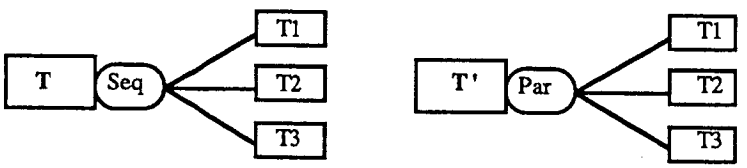


-a- -b-

Figure 2 : Décompositions temporelles d'une tâche sequence (-a-), d'une tâche parallèle (-b-)

L'exécution de la tâche T implique l'exécution de la tache T1, suivie de l'exécution de la tâche T2 et enfin de la tâche T3. La structure de la tâche T est de type séquence. L'exécution de la tâche T' implique l'exécution, dans un ordre quelconque, de la tache T1, de la tâche T2 et de la tâche T3. La structure de la tâche T' est de type parallèle.

- La seconde représentation graphique (Fig.3, Fig.4) de la tâche est un **arbre de décomposition logico-temporelle** qui exprime les décompositions successives de la tâches en sous-tâches de plus en plus fines. Les feuilles de l'arbre désignent des tâches simples c'est à dire des entités élémentaires insécables. L'axe représente le niveau d'abstraction de description de la tâche.



- a - - b -

Figure 3 : Arbres de décomposition logico-temporelle d'une tâche de type séquence (-a-) et d'une tâche de type parallèle (-b-)

La racine des arbres de décomposition (a- et b-) représente respectivement les tâches T et T'. Les noeuds fils de la racine expriment la décomposition de la tâche en sous-tâches plus spécifiques liées entre elles par un constructeur temporel. Les sous-tâches permettent d'affiner la description de la tâche.

exemple :

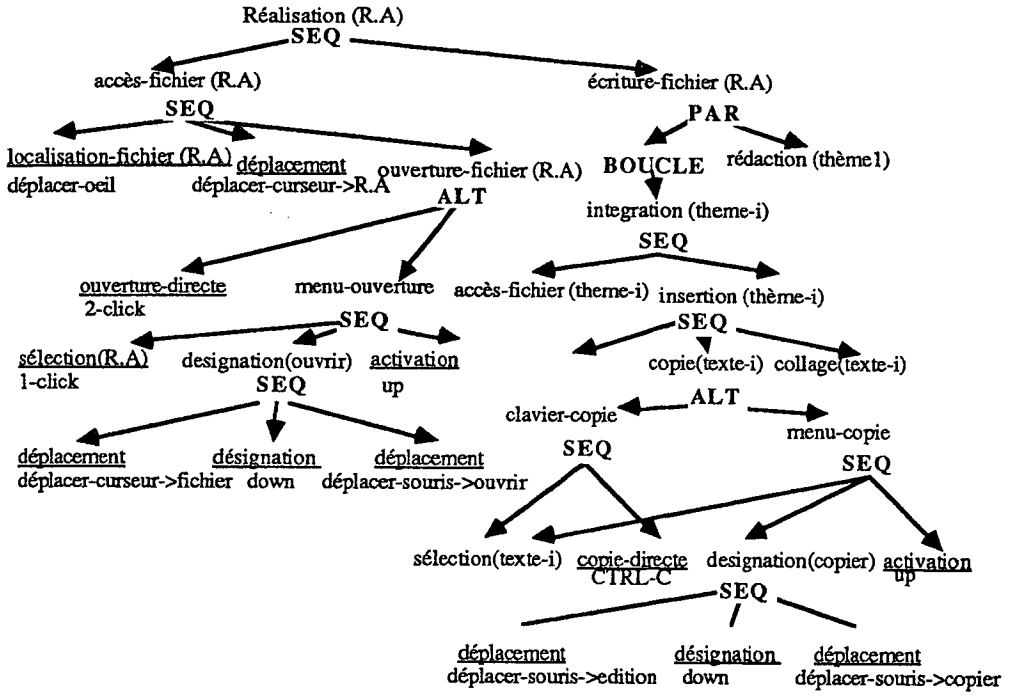


Figure 4: Graphe logico-temporel de l'activité "réalisation-rapport-activité"

Pour cet exemple, on a imaginé le scénario suivant: Il s'agit de l'activité de rédaction du rapport d'activité d'une équipe de recherche. On suppose que le rédacteur du rapport d'activité travaille sur un micro, a partir d'un fichier formaté fourni par la direction. Son futur rapport d'activité comporte 3 parties: la partie du rédacteur (le chef de projet), qui reste à rédiger (liste des membres de l'équipe, rédaction de son thème), et 2 parties déjà rédigées par deux chercheurs responsables chacun d'un thème et qui ont fourni une disquette décrivant respectivement leur activité. Le travail du rédacteur est donc essentiellement d'accéder à un fichier formaté, de rédiger sa partie, et d'incorporer au rapport deux textes a partir de disquettes.

4. L'implémentation du modèle

On distingue trois niveaux de représentation auxquels correspondent trois modules du système :

- Représentation externe et Interface: une interface réalisée en Aïda, permet à l'utilisateur de décrire dans le formalisme MAD une tâche, quelque soit son niveau de complexité. Il est ainsi possible, de décrire des tâches de haut niveau, c'est à dire des tâches faisant appel à un enchaînement structuré de sous-tâches.
- Représentation interne intermédiaire: une représentation interne intermédiaire est automatiquement générée à partir de ces informations, et traduite ensuite dans un formalisme objet.
- Représentation interne: il s'agit d'une représentation sous forme d'objets. Le modèle d'objets retenu est Shirka (Rechenmann, 1988). Certaines modifications du modèle ont été nécessaires.

L'avantage de l'utilisation d'un niveau interne intermédiaire est de permettre d'une part de distinguer le niveau interface du niveau application, d'autre part d'autoriser le passage éventuel à d'autres formalismes objets. La maquette en cours de développement peut déjà être utilisée à deux fins:

- d'une part, pour le concepteur de la base comme outil d'aide à l'acquisition des connaissances (dans la mesure où il permet de passer de la représentation que se fait l'utilisateur de sa tâche à la représentation de cette tâche dans un formalisme objet)

- d'autre part, pour l'utilisateur comme outil d'aide à la description de sa propre tâche.
Cet outil permet à la fois de créer une base de tâches prototypiques (catalogues des tâches usuelles) et de faire évoluer cette base en l'enrichissant, par la construction de nouvelles tâches. L'intérêt d'une représentation interne sous forme d'objets tels que les objets Shirka réside précisément dans le maintien de la cohérence de la base assuré par Shirka lors de la création de nouvelles tâches.

4.1 L'interface graphique

L'interface graphique (actuellement développée en AIDA et prochainement en MASAI) a été conçue pour permettre à un utilisateur de formuler simplement et interactivement la description de sa tâche mais aussi pour lui permettre de suivre et de contrôler l'évolution de la base de connaissances. Elle est donc constituée de deux modules:
- un éditeur de tâches pour la construction, la consultation et la modification interactives de tâches dans le formalisme MAD
- un éditeur de schémas pour la consultation de la base d'objets Shirka.

Le système d'acquisition et de représentation graphique des tâches en cours de développement, intègre les deux types de représentations graphiques présentées précédemment. La représentation logico-temporelle est utilisée pour l'acquisition de nouvelles tâches car elle permet de décrire complètement une tâche, en précisant les décompositions successives de la tâches en sous-tâches de plus en plus fines ainsi que l'agencement des sous-tâches pour chacune des décompositions. Un module de traduction sera implémenté pour permettre la consultation de la représentation temporelle d'une tâche quelconque de la base de connaissances.

4.1.1 Construction interactive de tâches

L'éditeur de tâches est composé d'un *éditeur d'arbre* qui permet à l'utilisateur d'expliciter interactivement la décomposition structurelle d'une tâche en sous-tâches, et d'un *éditeur d'objet*, permettant la consultation et la modification des informations caractérisant la tâche associée à chaque noeud de l'arbre.

Une tâche est représentée par le noeud racine d'une arborescence, ses sous-tâches par les noeuds-fils du noeud racine (cf 3.3 représentation logico-temporelle d'une tâche). Les feuilles de l'arbre définissent des tâches de type simple c'est à dire des composants élémentaires qui pointent sur une action à exécuter ou plus généralement sur un traitement procédural externe. L'*éditeur d'objet* permet à l'utilisateur de saisir les données caractérisant la tâche associée à chaque noeud de l'arbre, c'est à dire les informations concernant les objets en entrée et en sortie de la tâche, les pré-conditions et les post-conditions attachées à son exécution, le nom de l'action pour les tâches simples.

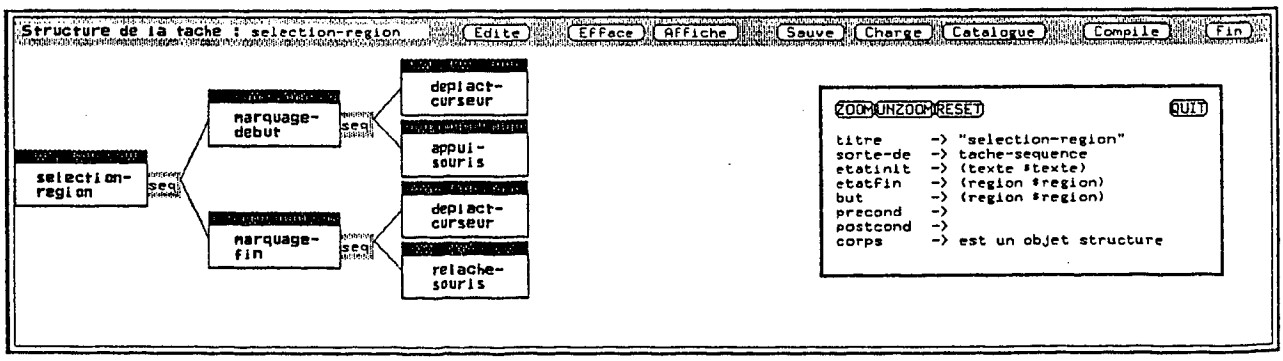


Figure 5 : Edition de l'arbre de la décomposition structurelle de la tâche Selection-région et du noeud racine de l'arbre.

Une structure intermédiaire modélisant la tâche dans le formalisme MAD, est construite dynamiquement à partir des informations contenues dans l'arbre. Cette structure, indépendante de l'environnement objet choisi, peut être compilée sous forme d'entités Shirka. Lors de la compilation les schémas Shirka nécessaires à la représentation objet d'une tâche, c'est à dire une classe spécialisation d'une classe tâche existante et l'instance prototypique de cette classe (cf. § 5) sont générés.

La compilation permet ainsi de passer de la représentation-utilisateur d'une tâche, exprimée dans le formalisme MAD, à une représentation orientée Objet. L'utilisation d'une structure interne intermédiaire permet d'une part de distinguer le niveau interface du niveau application et d'autre part de faciliter le passage à d'autres représentations objets.

Lorsque la structure intermédiaire a été compilée, l'objet tâche généré est intégré dans la base (catalogue des tâches) et peut être utilisée pour la construction d'une nouvelle tâche composée. La compilation permet donc l'extension de la base de connaissances.

La compilation d'une tâche composée est récursive c'est à dire qu'elle implique la compilation des sous-tâches qui ne figurent pas dans le catalogue des tâches existantes.

Exemple :

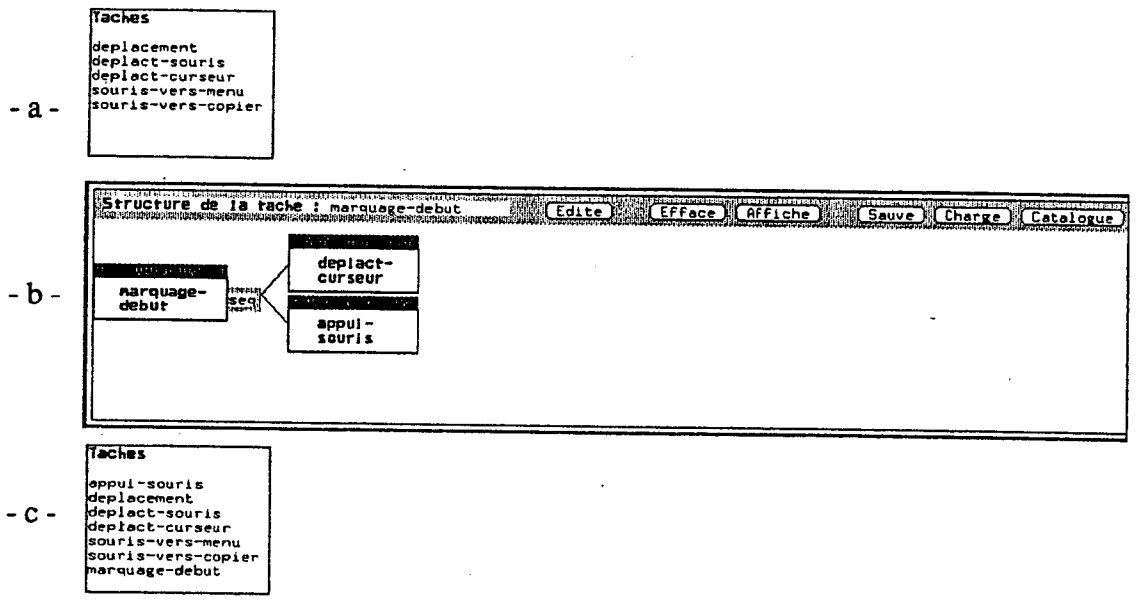


Figure 6 : Evolution de la base des tâche après compilation de la tâche Marquage-debut

Initialement, la base des tâches (a) ne contient que les tâches prédéfinies de haut niveau, la tâche Déplacement et ses spécialisations. Après compilation (b) de la tâche Marquage-debut (permettant de sélectionner le début d'une zone d'un texte), Marquage-début et Appui-souris sont intégrées dans le catalogue des tâches (c). Ce sont alors des entités indépendantes qui pourront être utilisées pour construire de nouvelles tâches composées.

4.1.2 L'éditeur de schémas pour la consultation de la base de connaissances

Les objets SHIRKA manipulés par l'application sont classés dans les quatre catégories suivantes : les **objets-tâches** modélisant les tâches prédéfinies de haut niveau et les tâches construites interactivement par un utilisateur, les **objets de l'application** (Fig. 7b) qui décrivent l'état courant du domaine d'étude et sont référencés dans la liste des objets en entrée et en sortie des tâches ainsi que dans les préconditions et les postconditions, les **objets-système** qui définissent environnement Shirka et en contrôlent la cohérence et les **meta-schémas** qui modélisent le niveau d'abstraction le plus haut, permettant de décrire le concept générique de tâche (et d'objet composite).

Un éditeur de schémas a été conçu pour autoriser la consultation d'un objet SHIRKA, quelque soit son type. Il se compose d'un éditeur d'arbre permettant d'afficher la hiérarchie de classes et d'instances associée à l'objet (L'utilisateur peut choisir de visualiser ou non les instances attachées aux classes intervenant dans la hiérarchie) et d'un éditeur de texte permettant, au niveau de chaque noeud de l'arbre, de consulter les informations caractérisant le schéma Shirka représenté par le noeud.(Fig 7a et 7b).

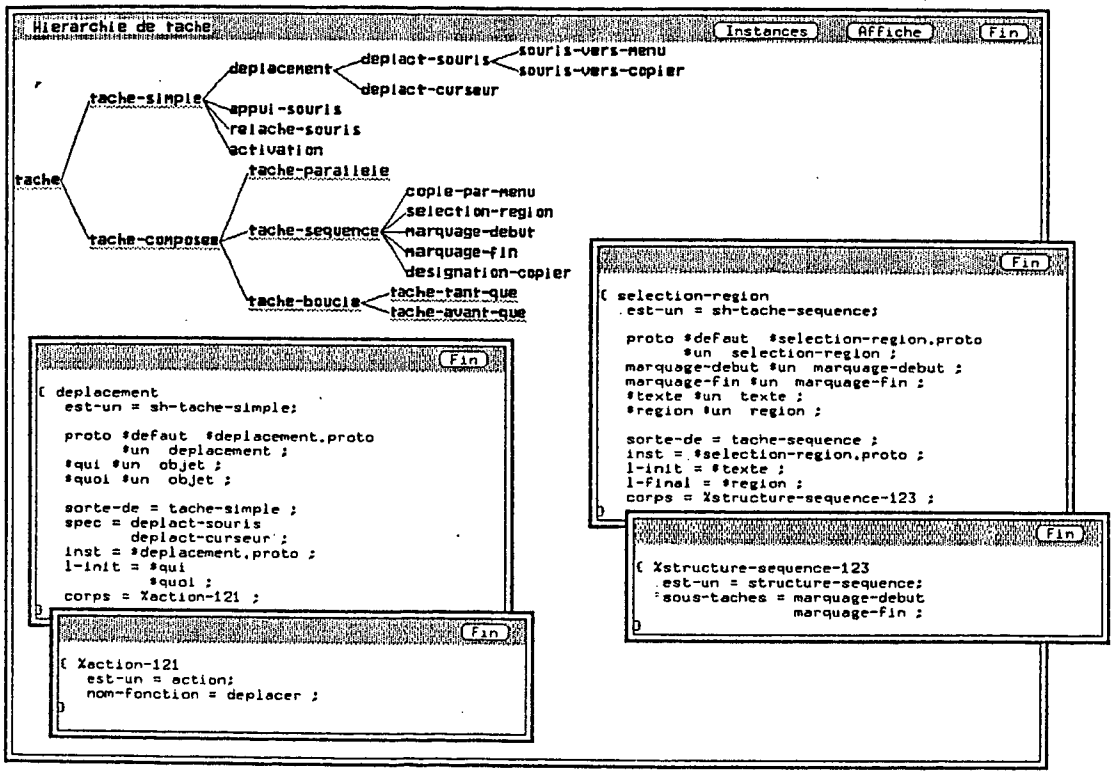
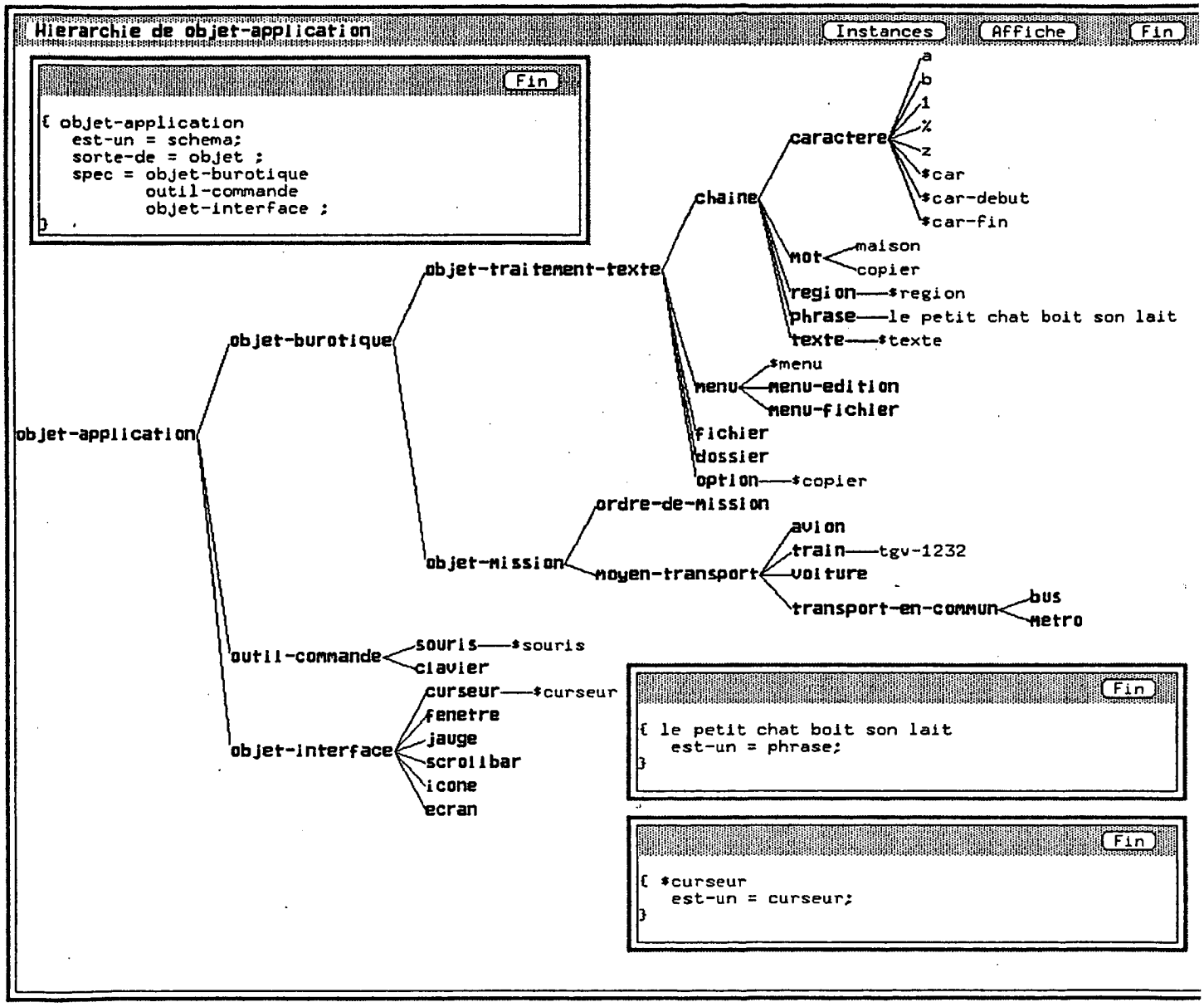


Figure 7a : Edition de la hiérarchie de classes et d'instances associée au schéma Tâche. Un éditeur de textes a été ouvert sur les noeuds *deplacement* et *selection-region*

L'éditeur de schémas permet le suivi et le contrôle de l'évolution de la base des tâches et du domaine d'étude modélisé par les **objets de l'application.**

Figure 7b : Edition de la hiérarchie de classes et d'instances associée
au schéma *Objet-application*



4.1.3 Consultation de la base des tâches

L'interface graphique permet d'afficher les tâches selon différents points de vue :

- L'éditeur de tâches permet de visualiser le graphe de décomposition logico-temporelle associé à une tâche. Ce graphe exprime les niveaux de décomposition successifs de la tâche en entités de plus en plus fines.
- Un module (non implémenté) permettra de consulter la représentation temporelle d'une tâche (cf §3.3). Cette représentation explicite l'enchaînement chronologique des différents traitements permettant d'exécuter la tâche.
- L'éditeur de schémas permet d'afficher la forme interne des objets-tâches manipulés par l'application. Il autorise notamment la consultation des meta-schémas décrivant le concept générique de tâche, des classes modélisant les tâches (Fig.7a, Fig.10) et des instances de ces classes simulant l' exécution des tâches.

4.2. Représentation objet des tâches

L'objectif visé est de représenter une tâche en terme d'objets, en partant d'un modèle de type frames "classique". Le modèle retenu à cette fin est Shirka. Cet objectif a conduit d'une part à une extension de ce modèle par la modification de son niveau meta et d'autre part à introduire ou préciser certains concepts tels que ceux d'instance générique, d'instance prototypique ou d'objet composite. L'intérêt d'une telle approche est de permettre d'appréhender à partir d'un problème réel (représentation des tâches) et d'un modèle objet existant, des questions fondamentales sur les modèles informatiques à base d'objets.

4.2.1 Introduction de meta-schémas pour la représentation des tâches

Le concept de tâche est représenté au plus haut niveau d'abstraction par un objet du niveau meta, le meta-schéma *sh-tache* , qui fournit une description générique de l'architecture de tout schéma *tâche*. En effet, une tâche, quelque soient ses spécificités, se caractérise par une liste d'attributs décrivant ses objets en entrée (l-init), ses objets en sortie (l-final), son but (but) et sa partie exécutive (corps). Par contre, aussi bien le nombre que la nature de ces attributs diffèrent selon les tâches: la liste et le type des objets de l'état-initial, de l'état-final, ainsi que la structure du corps de la tâche et son but dépendent de chaque tâche. Une tâche spécifique est donc représentée par un schéma de classe distinct, instance du meta-schéma *sh-tache*, décrivant le profil propre à cette tâche, c'est à dire la liste des ses entrées, la liste de ses sorties, la liste de ses sous-tâches etc.(Fig 10)

La définition du meta-schema *sh-tache* est ainsi la suivante:

```
{sh-tache
    sorte-de      =          Schema;
    l-init        $liste-de  Symbole;
    l-final       $liste-de  Symbole;
    but           $un        Symbole;
    corps         $un        Traitement}
```

L'introduction du meta-schema *sh-tache* permet pour chaque classe tâche, instance de *sh-tache*, de structurer ses attributs propres en distinguant les attributs représentant les paramètres d'entrée, les paramètres de sortie et la décomposition structurelle de la tâche. Toute instance de ce meta-schema est caractérisée par la valeur de ses champs l-init, l-final, but et corps. Les valeurs affectées à ces attributs permettent d'identifier les champs de ses instances (Fig.8).

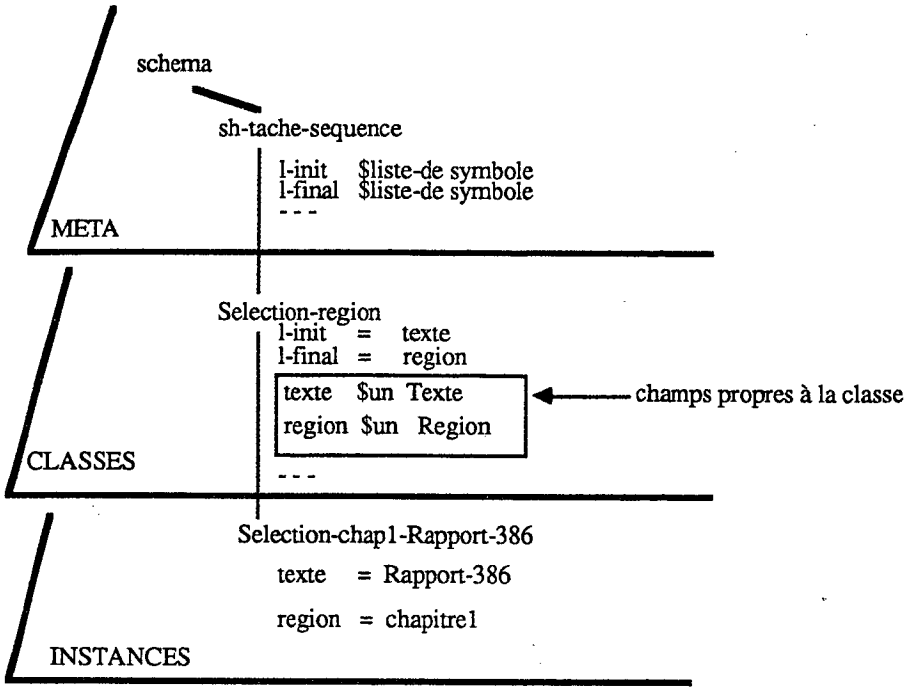


Figure 8 : Niveaux meta, classes, instances

Exemple:

La tâche Selection-region, décrite dans le formalisme MAD (§ 3), est représentée par le schéma selection-region, instance du meta-schema *sh-tache-sequence,* suivant :

```
{selection-region
    est-un          =     sh-tache-sequence
    sorte-de        =     Tache-sequence
    l-init          =     texte;
    l-final         =     region;
    but             =     region
    corps           =     {%structure-sequence1
                                constructeur  =     seq
                                sous-taches   =     marquage-debut marquage-fin}
    texte                 $un   Texte
    region                $un   Region
    marquage-debut        $un   Marquage-debut
    marquage-fin          $un   Marquage-fin
    ... }
```

La valeur 'texte' est affectée à l-init dans Selection-region, la valeur 'region' est affectée à l-final. Ces valeurs expriment que l' attribut texte de type Texte correspond à l'entrée et l'attribut region de type Region correspond à la sortie de la tâche Selection-region.

L'objet *sh-tache* est racine d'une hiérarchie de meta-schemas (Fig 9) qui permet de définir à son tour une hiérarchie de classes *tâche.*

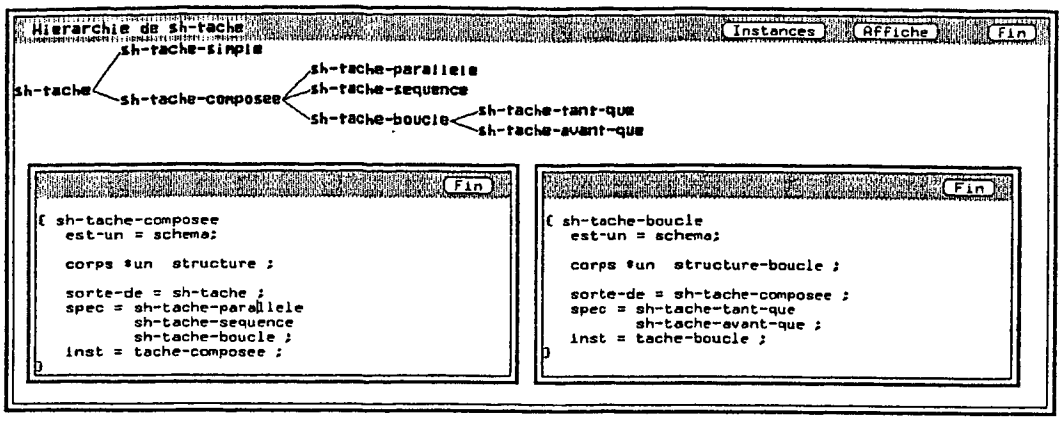


Figure 9 : Hiérarchie des meta-schémas de type tâche
Ces meta-classes se différencient selon le type de leur structure, ce qui permet d'organiser la hiérarchie des classes tâches

4.2.2 Le niveau classe

Une hiérarchie d'objets génériques représentant les tâches prédéfinies est organisée à partir de l'objet générique *tache* (Fig.10). Une tâche spécifique est représentée par une sous-classe, spécialisation de *tache*. Les tâches sont hiérarchisées selon la nature de leur structure et le type de leurs objets en entrée et en sortie. Par exemple, la tâche Deplacement-souris est définie comme une sous-classe de la classe Deplacement (préalablement définie comme spécialisation de Tache-simple). La classe associée à une tâche, permet de préciser le type des objets en entrée et en sortie de la tâche ainsi que sa décomposition structurelle en sous-tâches.

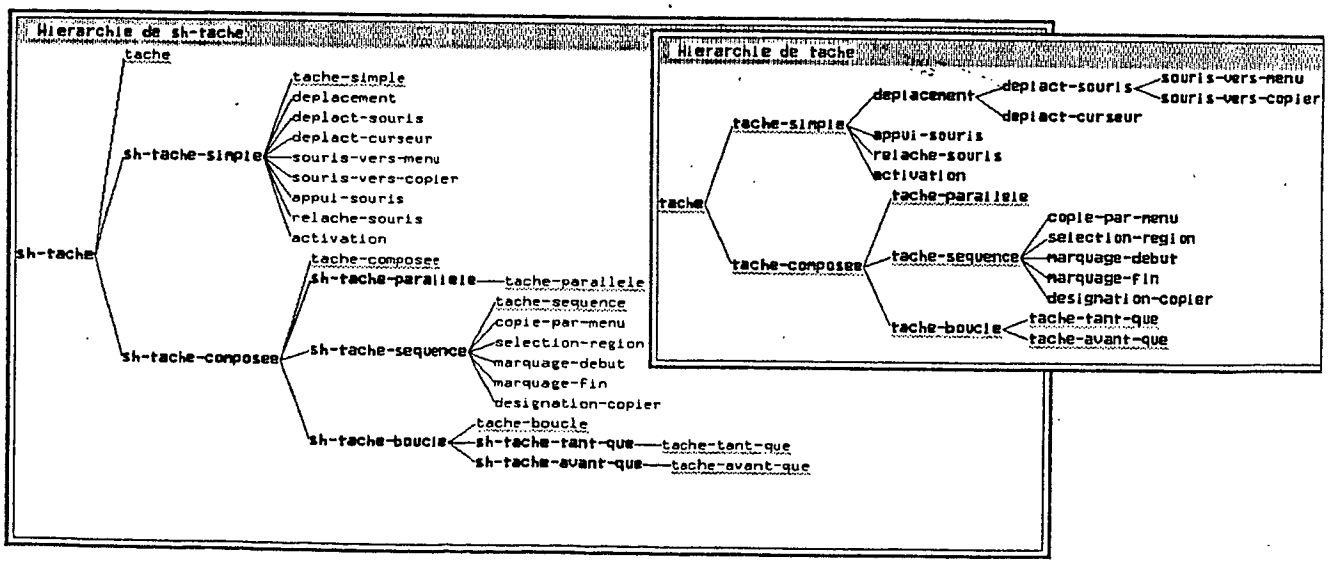


Figure 10 : Hiérarchie des tâches

Les tâches Selection-region, Marquage-debut etc. ont une structure de type séquence, elles sont donc représentées par des instances (en clair) du meta-schéma sh-tache-séquence (en gras), et sont des sous-classes de Tache-sequence, les tâches Appui-souris et Déplacement qui pointent sur des actions sont représentées par des instances du meta-schéma sh-tache-simple et sont des sous-classes de Tache-simple.

4.2.3 Le niveau instance

Une instance de la classe représente un cas particulier de réalisation de la tâche et constitue donc une mémoire de l'exécution de la tâche.

Exemple:

Soit une instance de la classe selection-region qui simule la sélection du premier chapitre du rapport 386:

```
{selection-chap1-Rapport-386
    est-un          =    selection-region
    texte           =    Rapport-386
    region          =    chapitre1
    marquage-debut  =    {marquage-debut-chapitre
                              est-un = marquage-debut     ... }
    marquage-fin    =    {marquage-fin-chapitre
                              est-un = marquage-fin       ... }}
```

Une tâche est donc représentée par une classe, chaque instance de cette classe modélisant une réalisation particulière de la tâche. La description présentée ci-dessus a dû toutefois être complétée par l'introduction d'un attribut supplémentaire, l'attribut *proto*, permettant d'attacher un représentant prototypique à une classe. En effet, la description précédente ne contient pas toute l'information utile à la définition d'une classe puisqu'en particulier elle n'explicite pas le passage de variables entre la tâche et les différentes sous-tâches. Par exemple, la tâche *Insertion* qui représente l'insertion d'une région d'un texte1 vers un texte2 a deux entrées texte1 et texte2, et admet comme sous-taches *Copie-par-menu* (copie d'une région du texte1), *Collage-par-menu* (collage de la région dans texte2). Pour assurer la cohérence de la description, il faut exprimer que la sous-tâche *Copie-par-menu* a pour argument d'entrée texte1 et que la sous-tâche *Collage-par-menu* admet pour variable texte2. Ce problème a été résolu en caractérisant chaque classe *tâche* par une instance prototypique. Comme dans d'autres domaines, l'introduction d'un représentant prototypique est un moyen commode de définir une classe dès lors que celle-ci représente un ensemble d'objets ayant une syntaxe particulière (voir par exemple, la définition de classes de modèles (PIERRET 88)). Les notions d'objet générique et d'instance prototypique de tâche sont des notions de portée générale, intéressantes dès qu'une classe doit être définie en termes de Modèle[2] satisfaisant une formule ou un ensemble de formules d'un langage donné.

4.3 La notion de variable universelle: instances génériques d'objet

Une instance générique d'objet est une instance d'une classe donnée qui fournit une description générique des autres instances de la classe. Elle permet de nommer une instance de la classe, ou en terme ensembliste de dire : soit texte1 un élément de l'ensemble Texte ou encore soit x un élément de l'ensemble Symbole. Une instance générique est un représentant abstrait des autres instances, elle correspond à la notion usuelle de variable universelle. Elle ne représente pas un objet concret mais un objet symbolique. Elle se distingue des autres instances par le fait que les valeurs affectées à ses attributs font référence à des objets génériques du type indiqué et non pas à des instances particulières. La définition d'une instance générique est donc la suivante:
- une instance générique d'objet simple est une instance quelconque de la classe des symboles
- une instance générique d'objet de type quelconque est une instance dont les valeurs d'attributs sont des instances génériques.

---

[2] Au sens de la logique

Exemples d'instances génériques:

soit la classe Texte:

```
{Texte
    sorte-de      =     Document;
    auteur        $un   Personne;
    date          $un   Date;
    nb-ligne      $un   Entier}
```

L'instance générique $texte1 est définie par :

```
{$texte1
    est-un        =     Texte;
    auteur        =     $personne;
    date          =     $date;
    nb-ligne      =     $n}
```

Par convention, le nom des objets génériques est toujours précédé d'un '$': $texte1, $personne, $date et $n représentent des instances génériques respectivement des classes Texte, Personne, Date et Entier.

Les objets symboliques $texte1 et $texte2 référés dans la tâche Insertion (dont la description en MAD a été donnée au § 3.1.3) sont représentés par deux instances génériques $texte1 et $texte2 de la classe Texte. Lors de l'instanciation de la tâche Insertion, elles pourront être substituées, par n'importe quelles instances de la classe Texte.

Exemple d'instance:

Soit mon-texte, une instance au sens classique de la classe Texte, on a :

```
{mon-texte
    est-un        =     Texte;
    auteur        =     Dupont;
    date          =     {date≠1
                              mois = Avril;
                              jour = 8;
                              annee = 1989}
    nbre-ligne    =     155}
```

4.4 Les instances prototypiques de tâche

L'instance prototypique d'une classe tâche est une instance générique particulière, attachée à la classe par l'intermédiaire du champ proto. Elle fournit un modèle qui permet de construire les autres instances de la classe par analogie et de reconnaître si une instance donnée de *tâche* appartient à cette classe.

Une tâche est ainsi décrite d'une part,
- par une classe qui exprime le type des objets en entrée et en sortie de la tâche, la décomposition structurelle de la tâche en sous-tâches, les préconditions et les postconditions attachées à son exécution
- d'autre part, par l'instance prototypique de cette classe qui décrit le passage des paramètres lors de l'enchaînement des sous-tâches et fournit un modèle permettant de construire les autres instances de la tâche par analogie.

Un couple (classe, instance prototypique) est donc automatiquement généré à partir de la description d'une tâche dans le formalisme MAD.

Exemple:

La tâche Insertion est définie en terme d'objets par :

- la classe Insertion, spécialisation de la classe Tâche-sequence

```
{ Insertion
    est-un    =    sh-tache-sequence
    sorte-de  =    Tâche-sequence
    structure =    { structure-sequence≠1
            constructeur  =     seq;
            sous-tâches   =     copie-par-menu
                                collage-par-menu }
    texte1                $un          Texte;
    texte2                $un          Texte;
    copie-par-menu        $un          Copie-par-menu;
    collage-par-menu      $un          Collage-par-menu;
    proto                 $un          Insertion;
                          $defaut      $Insertion.proto}
```

- L'instance prototypique de la classe Insertion, $insertion.proto :

```
{$insertion.proto
    texte1                =     $texte1;
    texte2                =     $texte2;
    copie-par-menu        =     {$copie-par-menu.proto
                  texte   = $texte1;
                  region  = $region
                  - - - }
    collage-par-menu      =     {$collage-par-menu.proto
                  region  = $region;
                  texte   = $texte2
                                ... } }
```

L'instance prototypique d'une classe tâche, comme toute instance générique, est caractérisée par le fait que les valeurs affectées à ses attributs sont des instances génériques des objets concernés.

C'est l'instance prototypique de la tâche qui permet de décrire le passage des variables lors de l'enchaînement des sous-tâches. Par exemple, dans le cas de la tâche *Insertion*, le prototype $insertion.proto indique que l'attribut texte de la sous-tâche *Copie-par-menu* doit être affecté avec la valeur $texte1 et que l'attribut region de la sous-tâche *Collage-par-menu* doit être affecté avec la valeur $region paramètre de sortie de la sous-tâche *Copie-par-menu.*

4.5 Spécialisations de tâches et hiérarchie de tâches prototypiques

4.5.1 Notion de spécialisation entre tâches

Certaines tâches sont des particularisations de tâches plus générales. Ainsi, on peut distinguer parmi les tâches exécutées à l'aide du menu d'un Macintosh, des tâches de différents types comme par exemple les tâches de manipulation de fichiers (ouverture, fermeture, enregistrement etc.) et les tâches d'édition (copie d'une zone, collage, effacement etc ...). Tandis que les premières manipulent des fichiers, les secondes manipulent des textes. D'autre part, les opérations auxquelles elles font respectivement appel appartiennent à des classes différentes, opérations d'édition et opérations de manipulation de fichier .

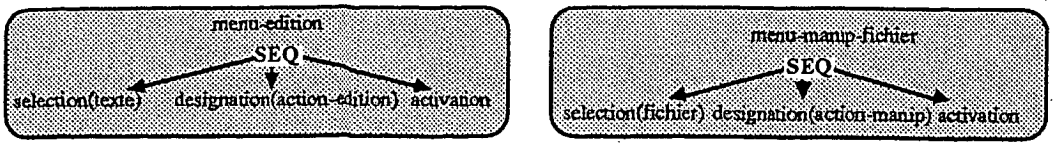


Figure 11: Décompositions structurelles des tâches *Menu-edition* et *Menu-manip-fichier*, spécialisations de la tâche *Menu*

Les tâches *Menu-edition* et *Menu-manip-fichier* explicitent les cas particuliers où les objets impliqués sont de type Texte ou Fichier. Ce sont des spécialisations d'une tâche plus abstraite (tâche *Menu)* décrivant la structure générique commune de toutes les tâches utilisant un menu (Fig.11).

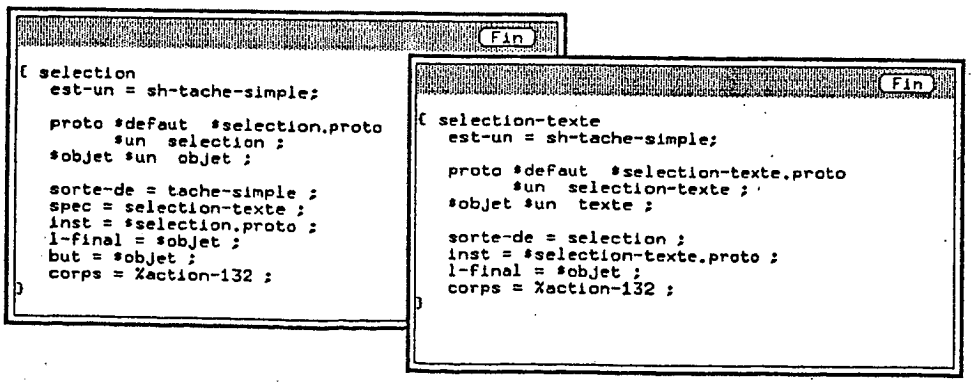
4.5.2 Définition

La notion classique de spécialisation entre objets appliquée aux tâches conduit aux conditions suivantes pour la spécialisation entre tâches : la spécialisation d'une tâche est un schéma possédant les mêmes attributs que la tâche de niveau supérieur (attributs hérités) et peut, d'autre part, faire intervenir de nouveaux attributs. Le type des attributs hérités est soit le même soit une spécialisation des types du niveau supérieur. En conséquence, une tâche T' est spécialisation d'une super-tâche T si:
- la tache T' possède les mêmes entrées et sorties que T , éventuellement augmentées d'un ou plusieurs objets. Dans le cas d'entrées (respectivement sorties) héritées, les objets en entrée (resp. sortie) de la tâche T' sont des spécialisations des entrées (respectivement sorties) de T.
- la tache T' possède les mêmes préconditions (resp.postconditions) que T éventuellement augmentées d'une ou plusieurs conditions. Dans le cas de conditions héritées, le type des prédicats de la tâche T' est une spécialisation de celui de T,
- la tâche T' possède le même constructeur et les mêmes sous-tâches que ceux définis dans T. Le type du constructeur et des sous-tâches est une spécialisation du constructeur et des sous-tâches de T. Si le corps de T n'est pas défini, T' peut faire intervenir un nouveau corps.

4.5.3. Exemples de spécialisation

- Cas d'une tâche simple

Soit la tâche *Selection* qui permet de sélectionner un objet de type quelconque affiché à l'écran. Cette tâche possède de nombreuses spécialisations telles que *Selection-fichier*, *Selection-texte*, *Selection-lecteur* qui spécifient le type de l'objet impliqué par le traitement. Les tâches *Selection* et *Selection-texte* sont caractérisées par les schémas suivants :



SHIRKA gère la cohérence en vérifiant que la classe Texte est une spécialisation de la classe Objet. Le corps de la tâche *Selection* , qui dans ce cas est de type action (Click-souris), est le même pour la classe et ses spécialisations.

- Cas d'une tâche composée avec spécialisation des sous-tâches

Soit la tâche *Menu* , précédemment mentionnée, qui permet d'activer une option d'un menu sur un objet préalablement sélectionné. Il lui est associé
- un graphe définissant sa décomposition structurelle et permettant d'affiner peu à peu sa description
- une hiérarchie de spécialisations permettant de travailler à différents niveaux d'abstraction (Fig.12).

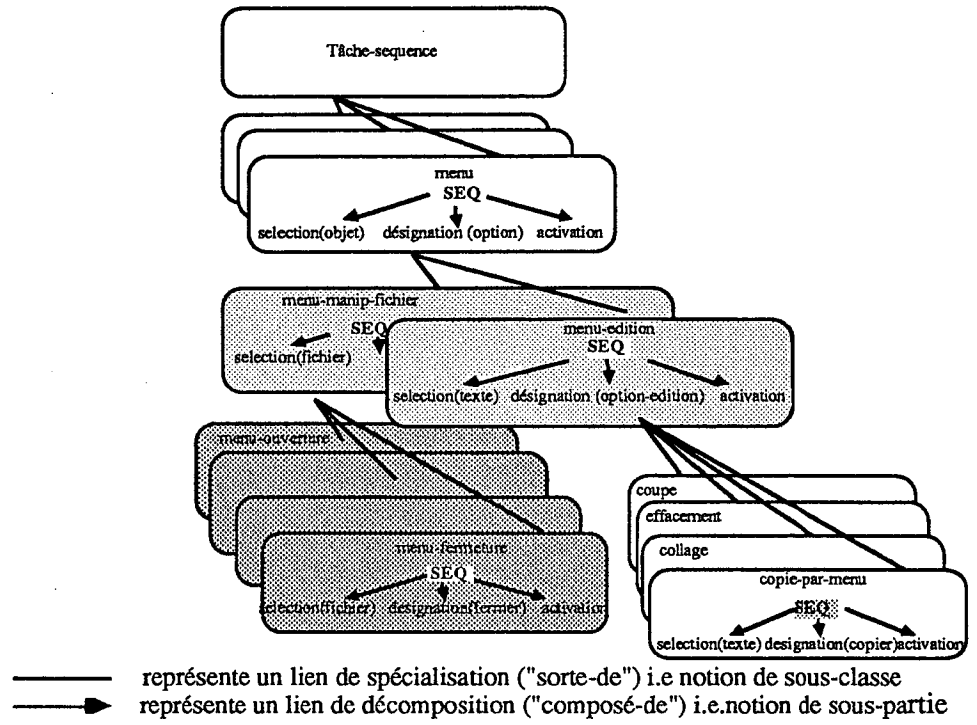


Figure 12 : description de la tâche *Menu*

La tâche *Menu* se décompose en trois sous-tâches de type : *Selection* (décrite précédemment), *Designation-option*, et *Activation*. La tâche *Menu-edition*, spécialisation de *Menu*, se décompose en trois sous-tâches de type plus spécifique : *Selection-texte* , *Designation-option-edition* et *Activation*. En effet, la tâche *Menu-edition* explicite le cas particulier ou le menu est utilisé sur un objet de type texte. Pour assurer la cohérence de la description, la sous-tâche selection de *Menu-edition* doit autoriser uniquement la selection d'un objet texte, c'est pourquoi elle est de type *Selection-texte* (spécialisation de la tâche *Selection*). De même, la sous-tâche désignation de *Menu-edition*, qui explicite le cas particulier ou l'option sélectionnée est une option d'édition, est de type *Designation-option-edition* (spécialisation de *Designation-option*). La spécialisation d'une tâche composée implique donc la spécialisation récursive de certaines de ses sous-tâches. Les tâches *Menu* et *Menu-edition* sont définies par les schémas suivants :

```
{Menu
    sorte-de        =       tâche-sequence
    l-init          =       (menu)
    corps           =       {structure-sequence≠123
                                    constructeur = seq
                                    sous-tâches =  selection design-option activation }
    menu            $un     menu
    selection       $un     Selection
    design-option   $un     Designation-option
    activation      $un     Activation ... }
```

```
{Menu-edition
    sorte-de        =       menu
    l-init          =       (menu)
    corps           =       {structure-sequence≠123
                                    constructeur = seq
                                    sous-tâches = selection design-option  activation }
    menu            $un     menu-edition
    selection       $un     Selection-texte
    design-option   $un     Designation-option-edition
    activation      $un     Activation ... }
```

L'attribut corps des tâches *Menu* et *Menu-edition* ont la même valeur (instance de Structure-sequence) car les deux tâches possèdent la même structure. Ces tâches se décomposent toutes les deux en trois sous-tâches permettant la selection de l'objet puis la désignation d'une option dans un menu et enfin l'activation de cette option sur l'objet. Elles diffèrent par le type de leurs objets en entrée et le type de leurs sous-tâches.

Les tâches peuvent être organisées en hiérarchie de niveaux abstraction. Au plus haut niveau d'abstraction, on peut trouver des tâches "abstraites", c'est à dire des tâches qui ne modélisent pas forcément des tâches réelles. Elles fournissent la description générique commune à toutes leurs spécialisations : nombre et type des objets en entrée et en sortie de la tâche, nombre et type des sous-tâches, etc ... (les types déclarés dans une tâche abstraite pouvant être spécialisées par ses sous-classes). Le corps d'une tâche abstraite peut ne pas être défini.

- Cas d'une tâche composée avec intervention d'un nouveau corps

Lorsque le corps d'une tâche abstraite est indéterminé, la partie exécutive de la tâche n'est définie qu'au niveau de ses spécialisations. Par exemple, la tâche Ouverture-fichier n'est pas associée à un traitement particulier, la procédure permettant d'ouvrir un fichier (dans le cadre de l'utilisation d'un Macintosh) étant dépendante de la méthode que l'utilisateur souhaite employer. Il peut faire un double-click sur le fichier ou utiliser le menu de manipulation des fichiers (menu-manip-fichier). La tâche Ouverture-fichier possède donc deux spécialisations, Double-click et Menu-ouvrir, qui spécifient les différents traitements permettant de l'exécuter. La tâche Ouverture-fichier est une tâche abstraite, pour l'exécuter il faut choisir l'une de ses spécialisations. Sur cet exemple le choix est fonction de la méthode retenue par l'utilisateur.
La tâche Ouverture-fichier est représentée par le schéma suivant :

```
{Ouverture-fichier
    sorte-de = tâche
    l-init = (choix-utilisateur  fichier)
    ...
    choix-utilisateur $un symbole
    fichier $un fichier
    corps = ( )
    proto = ... }
```

Les spécialisations Double-click et Menu-ouvrir sont définies par:

```
{Double-click
    sorte-de = Ouverture-fichier Tache-simple
    l-init = (choix-utilisateur  fichier)
    ...
    choix-utilisateur $valeur double-click
    fichier $un fichier
    corps = double-click
    proto = ... }
```

```
{Menu-ouvrir
    sorte-de = Ouverture-fichier  Menu-edition
    l-init = (choix-utilisateur fichier)
    ...
    choix-utilisateur $valeur lire-par-menu
    fichier $un fichier
    corps = {structure-sequence≠2
                    constructeur = seq
                    sous-taches = selection-fichier, designation-ouvrir, activation-ouvrir }
    proto = ... }
```

Pour exécuter une tâche abstraite, on sélectionne l'une de ses spécialisations selon le type des objets intervenant en entrée de la tâche (par un mécanisme de classification) puis on on exécute cette spécialisation.

- cas de spécialisation par ajout de préconditions ou de postconditions

On peut être amené à définir une spécialisation d'une tâche afin d'expliciter un cas particulier pour lequel les préconditions ou les postconditions doivent être complétées. Par exemple, la tâche Activation qui permet d'activer une option quelconque possède une spécialisation Activation-destruction-fichier qui effectue des vérifications supplémentaires (concernant les droits d'accès aux fichier, le type du fichier à détruire etc) avant d'exécuter le traitement d'effacement.

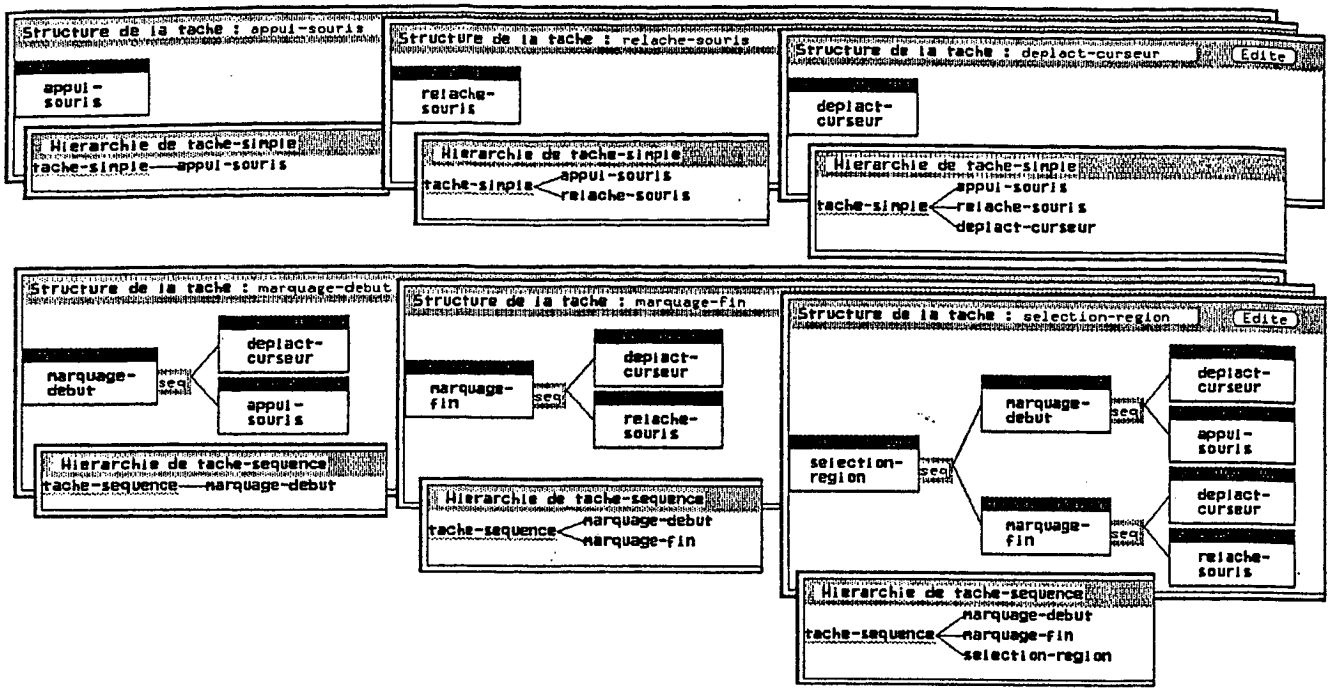
5. Un outil pour l'acquisition de connaissances

Le système en cours de développement est un outil d'aide à l'acquisition des connaissances. Il permet d'une part, la construction par l'expert d'un ensemble de tâches prédéfinies constituant la base de connaissances, et d'autre part, l'extension de cette base par un utilisateur, celui-ci pouvant définir ses propres tâches (Fig. 13).

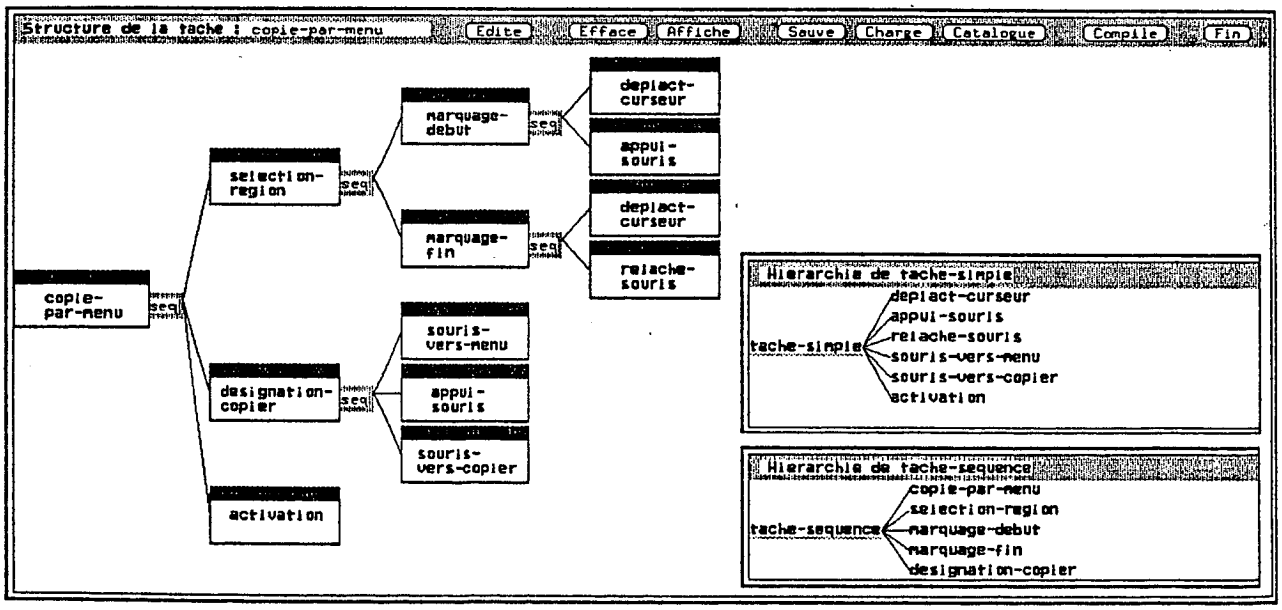
Comme on vient de l'exposer, le principe général d'utilisation du système consiste à décrire interactivement dans le formalisme MAD (cf §3) une tâche telle que l'utilisateur se la représente (au moyen de l'interface graphique). Cette description est alors compilée sous forme d'objets (schémas Shirka). La souplesse d'utilisation du système repose précisément sur l'utilisation d'un formalisme objet pour représenter les tâches. En effet, en procédant de la sorte les objets-tâches sont des entités **cohérentes et indépendantes.** Elles ne possèdent aucun lien statique vers les tâches de haut niveau dans la définition desquelles elles interviennent. Par contre, chaque objet-tâche gère ses sous-tâches par l'intermédiaire de son instance prototypique, en contrôlant les contraintes attachées à leur enchaînement. Une tâche donnée peut être ainsi être employée dans la description de traitements complexes distincts de plus haut niveau. Par exemple, les tâches Appui-souris et Déplacement-curseur sont des entités élémentaires qui peuvent être utilisées dans diverses tâches complexes représentant des activités d'utilisation d'un Macintosh.

Il est important de noter que le mode de construction d'une nouvelle tâche n'est absolument pas imposé à l'utilisateur. Pour décrire sa propre tâche, l'utilisateur peut :
- construire une tâche aussi bien par une démarche "top-down" que "bottom-up". Ceci autorise la construction incrémentale de tâches plus complexes à partir des composants existants.

Exemple :



- a -



- b -

Figure 13: Différentes constructions possibles de la tâche composite Copie-par-menu.

Initialement aucun des composants de Copie-par-menu n'existe dans la base,

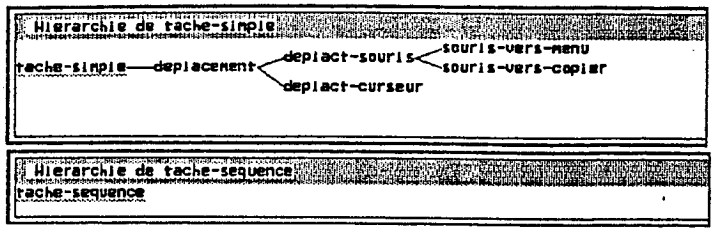
-a- démarche bottom-up : 1) création des tâches simples, Appui-souris, Relache-souris, Deplact-curseur
2) création des tâches séquences, Marquage-debut, Marquage-fin
3) création de la tâche séquence Sélection-region etc.
4) création de la tâche Copie-par-menu

-b- démarche top-down : création directe de la tâche Copie-par-menu

- combiner à la fois des tâches déjà définies dans la base et de nouvelles sous-tâches spécifiques. Chaque branche de l'arbre de décomposition logico-temporelle d'une tâche en cours de construction devient, après la compilation, une entité autonome et réutilisable.

Exemple :

- a-



- b -

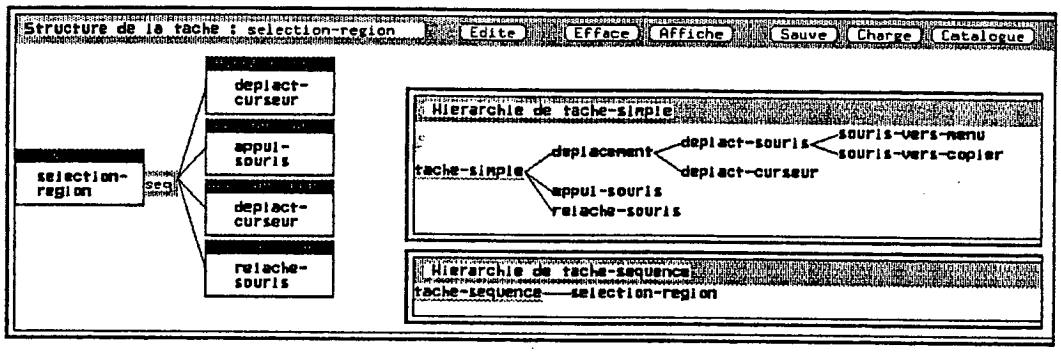


Figure 14: Evolution de la base des tâches

Initialement, la base des tâches ne contient que la tâche Déplacement et ses spécialisations (Fig. a). La tâche Selection est alors créée comme une combinaison de plusieurs sous-tâches: d'une part, la tâche Déplact-curseur qui existe déjà dans la base et d'autre part, les nouvelles tâches Appui-souris et Relache-souris. La compilation de la tâche Selection (Fig. b) modifie la base des tâches en générant la tache-sequence Selection-region et les tâches simples Appui-souris et Relache-souris (Fig. b).

- définir une nouvelle tâche par modification d'une tâche déjà définie dans la base. L'interface graphique permet en effet aisément l'ajout, la suppression de sous-taches ou l'introduction d'un niveau de décomposition intermédiaire pour une tâche existante (Fig. 15 - 16).

Exemples:

La tâche Selection-region était initialement définie comme composée des sous-tâches Déplact-curseur, Appui-souris, Déplact-curseur, Relache-souris (Fig. 14b). Elle a alors été modifiée par l'ajout d'un niveau de décomposition.supplémentaire : tâches Marquage-debut et Marquage-fin.

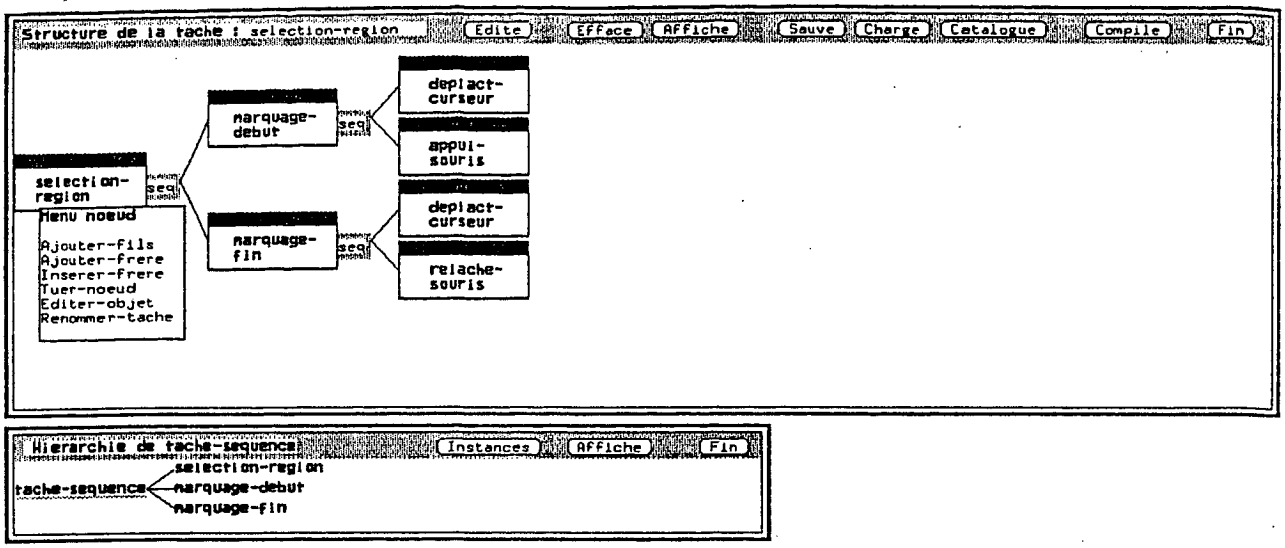


Figure 15: Modification de la définition d'une tâche
par introduction d'un niveau de décomposition intermédiaire

La modification se fait interactivement à l'aide du menu attaché à chaque noeud qui permet de couper, ajouter ou d'insérer un noeud fils. Les tâches séquences Marquage-debut et Marquage-fin sont désormais définies dans la base des tâches.

- Lorsqu'une une tâche possède une description proche d'une tâche existante, elle peut être construite à partir d'une copie de cette dernière.

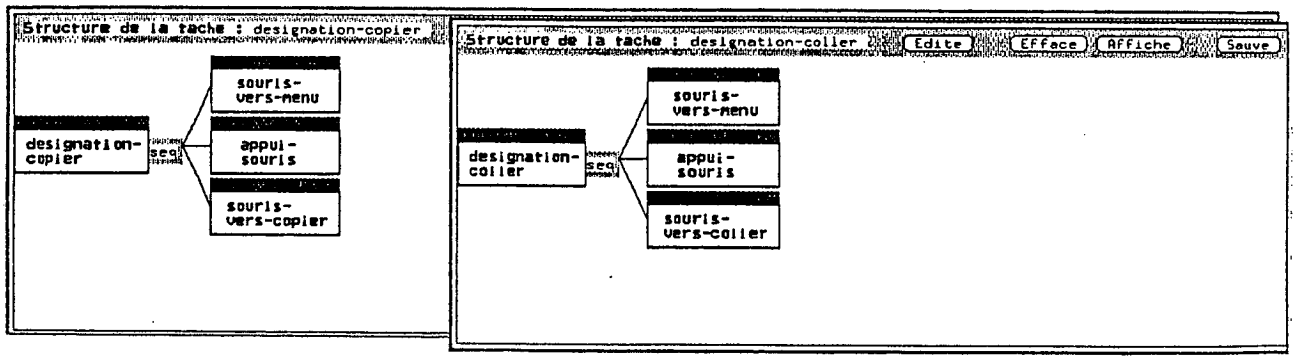


Figure 16: Création d'une tâche par modification de l'une des sous tâches
d'une tâche existante

La tache Designation-coller (qui permet de désigner l'option Coller dans un menu d'édition) est construite par copie de la tâche Designation-copier (permettant de désigner l'option Copier). La sous-tache Souris-vers-coller est substitué à la sous-tache Souris-vers-copier.

6. Les objets composites

Les notions de structure et de décomposition structurelle introduites précédemment à propos de l'objet *tâche* sont des notions de portée générale, utiles dans tous les problèmes concernés par des objets composites. Ce paragraphe est donc consacré à l'approfondissement du concept d'objet composite en vue de son implémentation dans un formalisme objet. Une extension des modèles objets classiques (ici Shirka) est proposée introduisant précisément ce concept dans le modèle. La définition d'un objet composite repose entre autre sur la notion d'instance prototypique.

Le concept d'objet composite s'avère aujourd'hui important dans différents domaines de l'Intelligence Artificielle. En effet, aussi bien les systèmes d'aide au diagnostic, à la conception que les systèmes concernés par l'apprentissage, montrent la nécessité d'introduire un tel concept.

- Si le diagnostic d'une situation peut souvent se formuler en termes de classification heuristique sur une base de prototypes (Clancey, 1983), tel n'est pas le cas lorsqu'il s'agit d'identifier une situation complexe (par exemple, situation simultanéité ou dépendance causale d'accidents, de maladies). Dans ce cas le problème ne peut plus être formulé en terme d'identification de la classe d'appartenance d'une instance parmi un catalogue de classes prototypiques. Il nécessite la définition d'une nouvelle classe, "modèle" de la situation complexe observée, reflétant la combinaison de modèles prédéfinis de la base (Patil, 1987).
- En conception également, il s'avère aujourd'hui important de pouvoir manipuler des objets composites. Le terme de conception, utilisé dans son sens le plus restrictif, renvoie à la définition donnée dans (Simon, 1983) : la conception en ingénierie consisterait à "préciser des objets sous contraintes" (i.e. à assembler des éléments connus choisis parmi un catalogue et à les dimensionner correctement). Mais il peut également renvoyer à un processus cognitif plus complexe où l'ingénieur est amené à inventer de nouveaux objets (i.e. la tâche ne se réduit pas seulement à un problème de configuration mais inclut le problème de la définition de l'architecture de l'objet). Tandis que ces dernières années ont vu se développer bon nombre de systèmes experts pour résoudre des problèmes de conception au sens de Simon, circuits VLSI (Mitchell, 1985), architecture Ili-Rise (Maher, 1985), mécanique (Brown, 1984), PRIDE (Mittal, 1986) configuration de systèmes informatiques R1 (McDephott, 1982), configuration de satellites EXSAT (Trousse, 1987), configuration de digues EXPORT (Neveu, 1987), un certain nombre de systèmes en cours de développement notamment en CAO, montrent la nécessité de s'attaquer au problème de la conception en son sens le plus général (Trousse, 1988). En effet, bien que dans les deux cas, configuration ou invention, la vocation de tels systèmes soit la même, i.e., de définir un objet de type composite satisfaisant certaines contraintes, le problème se formule toutefois de manière différente. Dans le premier cas, l'architecture de l'objet recherché est connue (une digue est composée d'un noyau, d'une carapace etc. (Neveu, 1987), une antenne est composée d'une structure-support (treillis), d'une surface réfléchissantes (tricot) etc., une aile de générateur est composée de n panneaux identiques équidistants ainsi que de divers mécanismes (Trousse, 1987). Ce type de problème se formule en termes de création d'instances et de caractérisation progressive de ces instances. Dans le second cas, l'architecture de l'objet est également à rechercher, le problème se formule alors en termes de création d'une nouvelle classe d'objets, aggrégation d'autres classes. Ce type d'applications soulève inévitablement au moins deux questions fondamentales interdépendantes: celle de l'évolution des bases de connaissances et de la définition d'objets composites reliés par d'étroites relations sémantiques. Comme le souligne par exemple B. Trousse (Trousse, 1988), il est fondamental en conception de "pouvoir, pour un assemblage modèle donné, spécifier un certain nombre de relations ou contraintes devant être vérifiées, par exemple pouvoir exprimer que tous les panneaux de l'aile-modèle sont équidistants d'une longueur d donnée". Ces problèmes se retrouvent à l'identique dans différentes disciplines comme en témoigne par exemple l'affirmation de Nguyen : "Conventional database systems have been simultaneously unable to support efficiently advanced applications, e.g. CAD/CAM and software engineering, which currently deal with large amounts of evolving and shared composite objects, which are linked by intricate semantic relationships " (Nguyen et al, 1988).

Dans le domaine des représentations objet en Intelligence Artificielle, la représentation des objets composites n'a été jusqu'ici que peu abordée ou peu approfondie. Parmi les systèmes s'étant penché sur cette question on peut citer LOOPS (Bobrow et al, 1983) et YAFOOL (Ducourneau et al, 1988). La raison en est peut-être que ce type de questions débouche rapidement sur des questions fondamentales concernant les modèles mêmes de représentation. Les paragraphes suivants rapportent l'état des réflexions conduites sur ce point à partir de l'application présentée précédemment.

Il est ainsi proposé d'introduire un meta-schéma *sh-composite* caractérisé par un champ *structure* permettant de spécifier les composants d'une classe d'objet composite et leurs relations.

6.1 Définition d'un meta-schéma *sh-composite*

La notion d'objet composite est introduite comme une extension du concept d'objet rendant possible l'instanciation d'un ensemble d'objets interconnectés, comme une entité. Un objet composite est donc un type d'objet particulier permettant de représenter des objets complexes structurés c'est à dire des aggrégations d'objets constituant une entité conceptuelle cohérente. Tout objet composite est caractérisé par un ensemble d'attributs définissant les propriétés de l'objet, un ensemble d'attributs définissant les composants de l'objet, et des attributs explicitant les relations entre ces divers composants. Tandis que les valeurs des propriétés des instances d'un objet composite dépendent de chaque instance, la structure des instances est commune. C'est une caractéristique de la classe. Lors de l'instanciation, seules les valeurs des composants diffèrent, leur type et leur relation sont identiques: deux instances d'un objet composite peuvent être considérées comme "isomorphes". Un objet composite peut référer à d'autres objets composites, auquel cas l'instanciation implique l'instanciation récursive de chacun de ses composants.

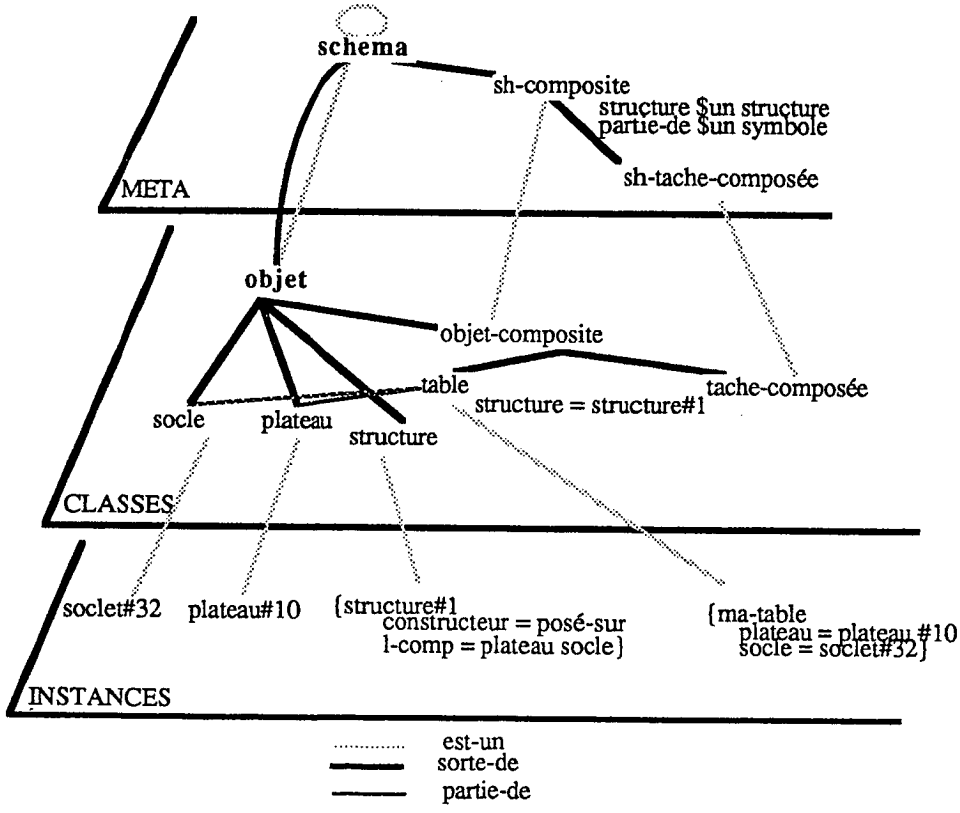


Figure 17 : Extension du modèle d'objets de Shirka par l'introduction du meta-schéma *sh-composite*

Exemple:

La classe Table est définie par:

- la liste des propriétés de ses instances: couleur, propriétaire, hauteur
- la liste de ses composants: plateau, socle
- la liaison entre ces composants: posé-sur

Le concept d'objet composite est donc représenté au plus haut niveau d'abstraction par le meta-schéma *sh-composite* possédant un attribut spécifique, l'attribut *structure*.

```
{sh-composite
    sorte-de        =           schéma
    l-att           $liste-de   attribut
    structure       $un         structure}
```

où structure est une classe définie par:

```
{structure
    sorte-de        =           objet
    constructeur    $un         constructeur
    l-comp          $liste-de   symbole}
```

De la sorte, l'attribut l-att, hérité de *schéma*, permet de spécifier l'ensemble des attributs caractérisant les instances d'un objet composite tandis que l'attribut l-comp permet de spécifier, parmi ces attributs, ceux qui représentent les composants. Le meta-schéma *sh-composite* enrichit donc le modèle objet en permettant de distinguer explicitement deux types de champs pour les objets composites. Cette distinction est importante puisque lors de la création d'une instance d'objet composite, les premiers attributs ne sont pas obligatoirement affectés alors que les seconds doivent nécessairement l'être.
Ainsi tandis qu'une classe d'objet est caractérisée par les attributs est-un, sorte-de, l-spec, l-inst et l-att permettant de définir respectivement sa classe d'appartenance, ses pères, ses spécialisations, ses instances, et ses attributs, une classe d'objet composite est caractérisée par ces mêmes attributs augmentés de l'attribut *structure* qui permet de décrire la décomposition structurelle de la classe. La structure est en effet une propriété caractéristique de la classe et non pas de ses instances. On introduit donc ici une différence fondamentale par rapport aux représentations habituelles des objets composites. En effet dans la plupart des systèmes les composants d'un objet-composite sont représentés par des attributs ordinaires, par exemple dans EXPORT une digue a pour attributs une carapace-ext, une carapace-int, un noyau, etc. Dans ERASME, un travail de type haut niveau a un champ sous-travaux qui indique que pour réaliser le travail de haut niveau concerné, il faut réaliser les différents travaux élémentaires mentionnés dans ce champ par leur nom de prototype (ERASME 88).

6.1.1 Structure

Un objet composite est donc caractérisé par une structure. La structure est un objet générique défini par un constructeur et une liste de composants. Le constructeur permet d'exprimer les relations ou contraintes d'assemblage entre les composants. On peut proposer un certain nombre de constructeurs prédéfinis exprimant des relations spatiales, temporelles, logiques, arithmétiques, etc.

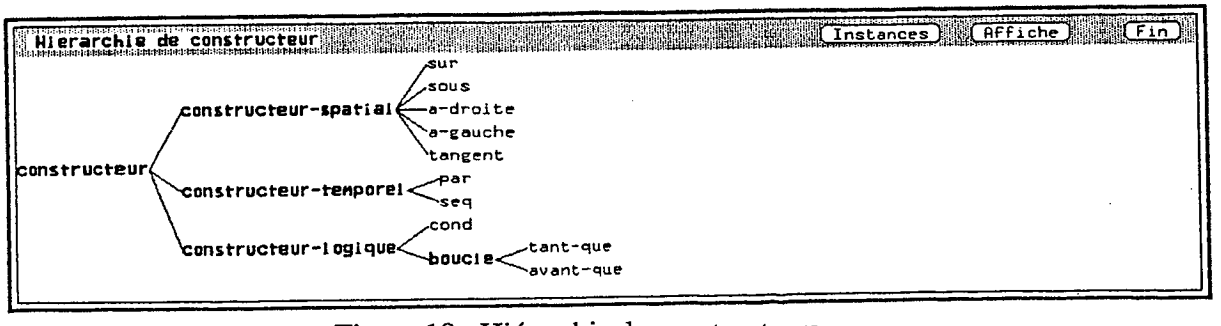


Figure 18 : Hiérarchie de constructeurs
Les classes sont figurées en gras, les instances en clair

6.1.2 Le lien *partie-de* et la notion de graphe de décomposition structurelle

De même qu'une hiérarchie de classes peut être construite à partir du lien *sorte-de*, lien de spécialisation, le lien structure permet d'introduire la notion d'arbre de décomposition structurelle. Un lien inverse de *structure* est introduit, le lien *partie de*, de façon à assurer la cohérence d'un graphe de décomposition structurelle. Il permet d'indiquer la position d'un objet dans ce graphe. Le lien *partie-de* est à *l-comp* ce qu'est *sorte-de* à *l-spec*.

6.1.3 Le niveau classe

Chaque type d'objet composite est représenté par une classe, instance du meta-schéma *sh-composite* ou de l'une de ses spécialisations. La valeur affectée au champ l-comp de cette instance permet d'identifier pour chaque classe la liste des champs représentants ses composants:

```
{Table
    est un          =       sh. composite
    sorte de        =       objet-composite
    l-att           =       couleur  propriétaire  hauteur plateau socle
    structure       =       {structure#1
                                    constructeur    =       posé-sur
                                    l-comp          =       plateau   socle}
    plateau         $un     Plateau
    socle           $un     Socle
    couleur         $un     Symbole
    propriétaires   $un     Personne
    hauteur         $un     réel}
```

Les composants peuvent être eux-même des objets composites, par exemple un socle peut être composé de quatre pieds équidistants

```
{Socle
    sorte-de        =       objet-composite
    partie-de       =       table
    l-att           =       matière  pied*4
    structure       =       {structure#2
                                    constructeur    =       équidistants
                                    l-comp          =       pied*4}
    pied1 $un pied
    pied2 $un pied
    pied3 $un pied
    pied4 $un pied
    matière $un matière}
```

6.1.4 Le niveau instance

La création d'une instance d'objet composite implique l'instanciation récursive de tous ses composants.

Exemple:

```
{ma-table
    est-un          =    table
    couleur         =    noir
    propriétaire    =    Personne#3
    plateau         =    plateau#32
    socle           =    {socle#10
                              est-un      =    socle
                              matière     =    bois
                              pied1       =    pied#12
                              pied2       =    pied#13
                              pied3       =    pied#14
                              pied4       =    pied#15} }
```

6.1.5 Intérêt de la notion d'instance prototypique dans la définition d'objets composites

Les instances d'une classe d'objet composite sont isomorphes, elles sont caractérisées par la même structure c'est-à-dire la même liste de composants, et le même constructeur. Ce sont les valeurs affectées à chacun des attributs (décrivant un composant ou une propriété) qui permettent de les différencier. Afin que la classe représentant un objet composite fournisse une description exhaustive de la structure commune de ses instances (cf § 6.1), une classe d'objet composite est définie à l'aide de l'une de ses instances, l'instance prototypique. Celle-ci est en effet nécessaire pour permettre d'expliciter toutes les contraintes caractéristiques de la structure modélisée. L'instance prototypique fournit un modèle des autres instances, c'est l'instance la plus générale que l'on puisse construire en respectant les contraintes entre les composants.

Exemples:

- Soit, par exemple, la classe des équations du second degré. Elle peut être représentée par un objet-composite, instance de sh-composite, exprimant qu'une équation du second degré est composée de trois monômes respectivement du premier degré, second degré et constant, reliés par un constructeur somme. Cette description est incomplète tant qu'on n'a pas précisé que ces trois monômes ont la même indéterminée. L'instance prototypique, ax^2+bx+c, de cette classe permet de donner une description complète de la classe en exprimant que l'indéterminée de chacun des monômes est la même: x. Elle fournit un modèle générique permettant de construire les autres instances. On peut noter que l'utilisation d'une telle instance prototypique pour définir une classe d'objet est un processus naturel: les équations du second degré sont naturellement décrites comme les équations de la forme ax^2+bx+c.

- Un triangle isocèle peut être considéré comme un cas particulier de polygone composé de trois côtés et représenté en terme d'objet composite par:

d'une part, une classe instance de *sh-composite* qui fournit la liste des composants et des propriétés d'un triangle isocèle :

```
{triangle-isocèle
    est-un          = sh-composite;
    sorte-de        = polygone;
    l-att           = surface
    structure       = {structure123
                            l-comp = (cote*3) }
    cote1           $un     segment
    cote2           $un     segment
    cote3           $un     segment
    proto           $un     triangle-isocèle
                    $defaut isocèle.proto
    surface         $un     reel}
```

où un segment est défini par:

```
{segment
    sorte-de        =       figure-plane;
    extrémité1      $un {point
                            xcoord          $un reel;
                            ycoord          $un reel}
    extrémité2      $un {point
                            xcoord          $un reel;
                            ycoord          $un reel}
    longueur        $un reel
                    - - - -                                 }
```

et d'autre part, une instance prototypique attachée à cette classe qui explicite les contraintes caractérisant un triangle isocèle:

```
{triangle-isocèle.proto
    est-un  = triangle-isocèle
    cote1   = {$segment1
                    extremité1      = $E1;
                    extremité2      = $E2;
                    longueur        = $x }
    cote2   = {$segment2
                    extremité1      = $E2;
                    extremité2      = $E3;
                    longueur        = $x}
    cote3   = {$segment3
                    longueur        = $y
                    extremité1      = $E3;
                    extremité2      = $E1}}
```

L'instance prototypique triangle-isocèle.proto pointe sur des instances génériques d'objets (cf §4.3) comme $x, $segment1 etc.

Une instance prototypique d'objet composite (comme par exemple, ax^2+bx+c) est un représentant abstrait mais cohérent de la classe qui permet de construire simplement d'autres instances de la classe ou de reconnaître si une instance appartient à cette classe. L'introduction de cette notion nécessite de développer un mécanisme de création d'instance spécifique aux objets composites, reposant sur la duplication de l'instance prototypique, ainsi qu'un mécanisme de mise en correspondance.
D'autres méthodes pourraient être utilisées pour définir de telles classes d'objet: par exemple, en ce qui concerne les équations du second degré, elle pourraient être définies comme somme de trois classes (monôme du second degré (ax^2), monôme premier degré (ay), etc.) avec un pointeur de y sur x, permettant d'indiquer que la valeur de y doit être la même que celle de x. La méthode proposée offre non seulement l'avantage du naturel mais apparaît d'une part moins lourde, d'autre part ne se limite pas à un artifice informatique quelconque permettant de respecter des passages de valeurs mais correspond bien à la notion de modèle.

Conclusion

Un formalisme de description des tâches a été proposé. Un modèle de représentation à base d'objets a servi de support à une première implémentation. On a montré comment ce choix apportait au système une grande souplesse d'utilisation et permettait d'assurer la cohérence des tâches ainsi représentées. L'outil présenté s'avère d'ors et déjà utile à l'extraction de connaissances procédurales et à leur représentation en termes d'objets.

Après avoir résolu un certain nombre de problèmes concernant la représentation explicite du concept de tâche, la perspective actuelle concerne maintenant les aspects dynamiques. Il s'agit notamment d'enrichir les modèles d'objets classiques par l'apport d'un mécanisme d'exploitation permettant d'étendre la notion habituelle d'attachement procédural. Ce mécanisme devra autoriser l'appel non seulement à des méthodes mais également à des tâches complexes pour inférer la valeur d'un attribut lorsqu'elle est nécessaire. A cette fin, un module de vérification de la cohérence des classes tâches, un mécanisme général de création d'instance sont en cours de réalisation

La formalisation objet du concept de tâche soulève toutefois certaines questions fondamentales sur les modèles objet : représentation explicite de connaissances procédurales et du raisonnement, évolution de la base d'objets, représentation des objets composites. La suite de ce travail permettra de continuer à approfondir sur le plan théorique la généralisation des solutions qui ont été proposées, ainsi que leur implémentation. Le cadre général de ce projet, i.e. la réalisation d'un système d'aide à la conception d'interfaces, s'avère adapté à la poursuite de la réflexion engagée sur ces thèmes. En particulier, la modélisation des objets de l'interface s'avère être un domaine intéressant pour approfondir et tester les idées avancées à propos des objets composites. La formalisation du raisonnement de conception d'interfaces sera l'application privilégiée utilisée pour définir une extension du formalisme de tâche, adaptée à la modélisation du raisonnement.

Bibliographie

Aida (1987) *AIDA : Environnement de développement d'applications,* ILOG, Paris 1987

Bobrow, D.G. (1983).Stefik M., *The LOOPS manual,* Xerox Corporation, 1983

Brown, D.C. (1984), Chandrasekaran B., *Expert system for a class of mechanical design activity,* IFIP Conference on Knowledge engineering in CAD, Budapest, Hongrie 1984

Chandrasekeran, B. (1987) *Towards a Functional Architecture for Intelligence Based on Generic Information Processing Tasks..* IJCAI Vol 2,1987.

Clancey, W.J. (1983) *Heuristic Classification* Artificial Intelligence, 27, 1985, pp 289-350

Dieng, R., Trousse B. (1988) *3DKAT, a dependency-driven dynamic-knowledge acquisition tool. Third Symposium on Knowledge Engineering,* Madrid espagne, 1988.

Ducourneau, R. (1988) *YAFOOL: Yet Another Frame-based Object Oriented Language,* Manuel de référence, Sema.metra, décembre 1988

Maher, M. et Fenves, S. (1984) *III-RISE: an expert system for the preliminary structural design of high rise buildings,* IFIP Working group 5.2,IFIP Conference on Knowledge engineering in CAD, Budapest, Hongrie 1984

Mitchell, T. et Shulman, J. (1985) *Vexed, a knowledge based VLSI Design Consultant,* July 1985, Rutgers Technical Report LCSR-TR 57

Mittal, S. et Dym, C.L., Morjaria M. (1986) *PRIDE: an expert system for the design of paper handling system,* IEEE 1986, pp102-114

Mittal, S. and Araya M. (1986) *A knowledge-based framework for design,* 5th National Conference on Artificial Intelligence, Philadelphia, 1986.

Neveu, B. (1987) *EXPORT: an expert system in breakwater design,* ORIA 87, Artificial Intelligence and sea, 11-17, 1987.

Nguyen, G. et al, (1988) *Schema Evolution in object-oriented database systems,* Rapport de Recherche INRIA N° 947

Patil, S. (1987) *A Case Study on Evolution of System Building Expertise: Medical Diagnosis.*, A.I. in the 1980s and Beyond, An MIT Survey, Grimson W. Eric L. and Patil Ramesh S.(Eds.), The MIT Press, Massachussets, 1987.

Pierret, C. (1988)*Vers un système à base de connaissances centrées objet pour la modélisation de systèmes dyamiques enbiologie,* Thèse de doctorat de l'Université de Compiègne, novembre 1988.

Rechenmann, F (1988) *SHIRKA : Systeme de gestion de bases de connaissances centrées-objet.* Manuel d'utilisation 1988

Scapin, D.L. (1988) *Vers des outils formels de déscription des tâches orienés conception d'interface.* Rapport de Recherche INRIA 893, 1988

Scapin, D.L., Reynard P. and Pollier A.(1988). *La conception ergonomique d'interfaces: problèmes de méthode.* Rapport de Recherche INRIA 957, 1988

Scapin, D. and Pierret, C. (1988) *Une Méthode Analytique de Description des tâches* , Colloque sur l'ingénierie des interfaceshomme-machine, Mai 1989 , Cargèse

Smeci, (1988) *SMECI Version 1.3*, Manuel de référence", ILOG, Paris 1988

Trousse, B. (1987) *EXSAT: système expert configurateur de satellites de télécommunications,* Septièmes journées internationales AVIGNON 87, Tome 1 335-350, Avignon, France, May 1987.

Trousse, B. (1988) *Couplage IA et CAO pour le prédimensionnement de satellites,* Technospace, Bordeaux, France, December 1988.

Tueni, M., Li , J. and Fares, P. (1988) *AMS: A knowledge-based approach to tasks representation, manipulation and organization,* COIS-88, Palo Alto.

Imprimé en France
par
l'Institut National de Recherche en Informatique et en Automatique