Expression d'un modèle de tâches à l'aide d'une représentation par objets

Jérôme Gensel
gensel@lifia.imag.fr

Pierre Girard
girard@lifia.imag.fr

Projet SHERPA — IRIMAG-LIFIA
Bâtiment HITELLA
46, avenue Félix Viallet
38031 GRENOBLE CEDEX
Tel: (33) 76 57 45 80
Fax: (33) 76 57 46 95

Résumé:

L'intégration dans un système de représentation de connaissances par objets d'un modèle de raisonnement basé sur la notion de tâche augmente la déclarativité du raisonnement et doit permettre une meilleure exploitation de sa trace. Le modèle de tâches décrit met à profit la notion d'objet composite et les mécanismes d'instanciation et de classification du modèle de représentation par objets, TROPES, dans lequel il est intégré. Les tâches sont distinguées par les buts qu'elles permettent d'atteindre et par la diversité des traitements qu'elles proposent à ces fins. La modélisation d'une tâche et la production d'une trace de son exécution à l'aide d'objets composites, donc d'entités à part entière du modèle, pourront alors être exploitées, notamment pour l'explication du raisonnement, par les mécanismes de TROPES.

Mots clés : Tâche, objet composite, instanciation, classification, exécution de tâche, trace d'exécution.

Catégorie de l'article : technique

1. Introduction

Les systèmes qui visent la résolution de problèmes complexes ont à mettre en place des raisonnements élaborés qui tendent à reproduire la démarche d'un expert du domaine traité. Pour attribuer un caractère général à ce type de système, la modélisation du raisonnement s'avère alors nécessaire.

Modéliser le raisonnement consiste à fournir des outils permettant d'exprimer les différentes étapes et les connaissances requises par tout processus de résolution de problèmes. Dans cette optique, la notion de tâche est un outil puissant permettant d'exprimer la stratégie à suivre pour résoudre un problème donné. Cette notion est au coeur de méthodologies d'analyse de problèmes qui décomposent un problème en sous-problèmes et sont utilisées pour concevoir des applications [Stee90]. Dans ce sens, l'approche des tâches génériques proposées dans [Chan87] définit des modules spécialisés (tâches génériques) pour la résolution de types spécifiques de problèmes; ces tâches génériques sont alors les briques de base de la conception d'un système à base de connaissances. Ces deux propositions montrent l'intérêt de la prise en compte de la notion de tâche dans la conception d'une application. Toutefois, elles restent abstraites et n'offrent pas de structure concrète pour définir une tâche et son fonctionnement. Le modèle de tâches décrit ici, prend lui aussi en compte l'idée de décomposition de tâches en sous-tâches. Il est, de plus, exécutable et permet de décrire les tâches dans un formalisme unique et d'en gérer les exécutions suivant une structure d'inférence puissante inspirée de la classification heuristique [Clan85] (adaptation au contexte particulier d'utilisation, puis exécution du traitement associé à ce contexte, à l'aide d'une décomposition en sous-tâches).

L'intégration de tâches au sein d'un modèle à objets aboutit à une représentation déclarative et homogène des connaissances du domaine et des connaissances stratégiques. Par exemple, le modèle proposé par Smith [Smit&84] représente des tâches par des objets et permet la décomposition d'une tâche en sous-tâches. Cependant, il ne permet pas l'adaptation d'une tâche au contexte. Par contre, le modèle de tâches SCAI [Ponc&91], qui est intégré au système de représentation par objets SHIRKA [Rech&89], exploite la classification d'instance pour diriger l'exécution des tâches en fonction des données. Diverses utilisations de SCAI, notamment en biologie moléculaire [Uvie&91], valident l'intérêt de cette approche.

Le principe d'intégration d'un modèle de tâches dans une représentation par objets est ici repris avec un accent particulier sur la production d'une trace d'exécution. L'intérêt d'une telle trace est reconnu, que ce soit dans le domaine de l'apprentissage où les traces sont un ensemble d'exemples à généraliser, dans le domaine de l'explication du raisonnement [DAVI&90], [HASL&84] où une trace est exploitée pour donner l'explication d'un raisonnement particulier ou bien encore dans le domaine de la reconnaissance d'intention [NAUL&91] où l'intention est assimilée à une trace que le système essaie de reconnaître.

La trace d'exécution d'une tâche est décrite par un réseau d'objets dont chaque noeud, en tant qu'objet, peut être manipulé et exploité par les différents mécanismes de la représentation par objets sous-jacente. Pour construire ce réseau, il convient de s'intéresser plus particulièrement à la modélisation du lien entre une tâche et ses sous-tâches, c'est-à-dire l'ensemble des tâches qui la composent et que l'on peut observer à un niveau plus élevé de description. La tâche est ici représentée comme un objet complexe dont les sous-tâches sont des composants. Cet objet, composé d'éléments de même type que lui, a donc une structure récursive. La structuration des tâches en une hiérarchie de classes, qui obéit à l'organisation taxonomique d'une base de connaissance représentée par des classes et leurs instances, vise également à exprimer les différentes possibilités de décomposition d'une tâche en sous-tâches.

Le modèle TROPES, dont les principales caractéristiques sont brièvement présentées, permet d'exprimer de tels objets et offre les mécanismes nécessaires à leur exploitation. L'exécution de tâches y est assimilée à la notion d'instanciation d'objets composites.

2. Présentation du modèle de tâches

Un modèle de tâches doit permettre de définir le concept de tâche (offrir la représentation de ce concept), d'exprimer la décomposition d'une tâche en une séquence équivalente de sous-tâches (donner les outils de description des liens entre la tâche et l'ensemble de ses sous-tâches) et l'ordonnancement des sous-tâches, de lancer l'exécution de l'ensemble des procédures qui traduisent l'aspect calculatoire de la tâche (se charger de l'aspect dynamique de l'exécution).

2.1. Déclaration de tâche

A toute tâche est associé un but que l'exécution réussie de la tâche permet d'atteindre. La tâche regroupe un ensemble de traitements. Chacun de ces traitements décrit un moyen d'atteindre le but fixé de la tâche. Un traitement se caractérise par la description :

- de ses entrées, ou **contexte d'utilisation de la tâche** c'est-à-dire les valeurs admises pour lancer l'exécution de la tâche selon ce traitement particulier,
- de ses sorties, c'est-à-dire les valeurs attendues comme résultats et,
- du traitement lui-même, c'est-à-dire de la décomposition, propre à ce traitement particulier, de la tâche en sous-tâches.

La décomposition d'une tâche en sous-tâche, donnée dans la description du traitement associé, se poursuit au niveau des sous-tâches dans leur propre description, etc. Cette décomposition est donc récursive; elle s'achève par une tâche dite terminale associée à une procédure. Au plus bas niveau de décomposition d'une tâche selon un traitement donné, un ensemble de procédures (tâches terminales) est obtenu. L'exécution de cette tâche correspond aux exécutions en séquence de chacune de ces tâches terminales dans l'ordre induit par la décomposition de la tâche. Si les valeurs fournies par l'exécution de la dernière procédure se révèlent être compatibles avec la description des sorties de la tâche, l'exécution de la tâche est considérée comme réussie; ces valeurs devenant effectivement les sorties de la tâche.

Une **tâche concrète** est une tâche qui ne présente qu'un seul et unique traitement pour atteindre un but concret. Dans la description de ce traitement, la tâche se décompose en sous-tâches. Une **tâche concrète terminale** est une tâche indécomposable; elle décrit alors uniquement l'appel à une procédure. Une **tâche abstraite** est une tâche qui regroupe :

- des tâches abstraites dont le caractère est moins général (qui peuvent décrire un même but concret ou des buts concrets différents[1]) ou bien,
- des tâches concrètes ayant le même but concret.

Une tâche abstraite a un contexte d'utilisation plus général que le contexte de toute tâche abstraite ou concrète qu'elle regroupe. On obtient ainsi une **hiérarchie de spécialisation de tâches** en fonction du contexte d'utilisation; le sommet de cette hiérarchie est la tâche abstraite de contexte d'utilisation le plus général.

Par exemple, une hiérarchie de tâches peut avoir comme tâche abstraite la tâche "manipulation de matrices" (plusieurs buts et plusieurs traitements) qui se spécialise en deux tâches abstraites "inversion" et "multiplication" (chacune ayant un but et plusieurs traitements) qui peuvent être chacune résolues par des tâches concrètes (un but et un traitement). La sélection se fait dans un premier temps au

[1] l'union de ces buts concrets différents correspond au but de la tâche abstraite, qu'on appelle but abstrait

niveau des tâches abstraites sur le but à atteindre—par exemple, si le but est d'inverser une matrice plutôt que de la multiplier par une autre, la tâche "inversion" est choisie parmi les sous-tâches abstraites de "manipulation de matrice"—, avant de se faire sur le contexte d'utilisation (les entrées de la tâches)—par exemple, le choix de la tâche "inversion d'une matrice réelle" si les éléments sont tous réels.

2.2. Exécution de tâche

L'exécution d'une tâche reproduit la stratégie de résolution qui coordonne l'exécution des procédures.

La première phase d'exécution d'une tâche consiste à parcourir la hiérarchie de spécialisation afin de déterminer la tâche concrète correspondante à la situation d'entrée, c'est-à-dire le traitement à effectuer. La deuxième phase, qui intervient après la détermination du traitement, provoque l'exécution des sous-tâches (et donc la répétition de ce processus jusqu'à l'exécution d'une tâche terminale) d'après l'ordre induit par la décomposition.

L'exécution se termine lorsque :

- la tâche échoue :
  - une tâche concrète n'a pu être choisie,
  - toutes les tâches concrètes sélectionnées ont échoué, pour chacune d'elles, soit :
    - le traitement associé (l'exécution des sous-tâches) a échoué,
    - le traitement a abouti mais les valeurs rendues par l'exécution de la tâche ne sont pas valides au regard des contraintes fixées dans la description des sorties de la tâche.
- la tâche réussit :
  - le traitement a abouti et les contraintes sur les sorties sont vérifiées.

Le modèle de tâches présenté identifie l'information contenue dans une tâche, introduit les notions de décomposition et spécialisation de tâches et propose un mécanisme d'exécution tirant partie de ses éléments. Dans la section suivante, les éléments de représentation (concept, classe, instance, objet composite,…) et les mécanismes d'inférence (classification, instanciation,…) du système de représentation de connaissances par objets TROPES, vont permettre d'exprimer de façon déclarative les différentes notions du modèle de tâches (Cf. §4 et §5).

3. Présentation de TROPES

3.1. Systèmes de représentation de connaissances par objets

Un système à bases de connaissances doit, à la fois, permettre de représenter et de stocker une grande quantité de connaissances, mais aussi et surtout offrir des mécanismes de raisonnement qui pourront être appliqués sur les connaissances déjà acquises afin d'en obtenir d'autres.

Lorsque ces structures de données qui accueillent la connaissance et les méthodes d'exploitation qui définissent le contrôle sont imbriquées, la représentation de la connaissance est dite *procédurale*. La connaissance est ici autant dans les données que dans l'association des structures de contrôle qui forme le programme. Lorsque, au contraire, ces structures sont décrites indépendamment du contrôle, la représentation de la connaissance est dite *déclarative*. L'avantage d'une telle représentation est qu'il est alors possible de faire évoluer indépendamment les connaissances et les mécanismes d'exploitation (la partie procédurale) du système.

Le terme de "systèmes de représentation de connaissances basés sur des objets" (proches des "frames" [Fike&85]) , est souvent l'"objet" d'une confusion. En effet, la distinction entre les systèmes de représentation de connaissances basés

sur des objets et les systèmes de programmation orientée-objet utilisés pour représenter la connaissance n'est pas toujours faite, et il convient ici de la rappeler.

Lorsqu'on utilise un système de programmation orientée-objet tel que SMALLTALK[Gold&83] ou LOOPS[Bobr&81], faits et contrôle ne peuvent être qu'intimement liés sous forme de programmes. Comme le souligne Patel-Schneider [Pate90], les systèmes de programmation orientée-objet, bien qu'ils soient souvent utilisés pour représenter la connaissance, ne sont pas, par nature, réellement appropriés à cette tâche. Ils ne peuvent que fournir une représentation procédurale de la connaissance. Ils sont trop souvent amenés à mélanger l'information destinée à la représentation des connaissances à une information concernant des détails d'implémentation qui ne concerne donc pas la représentation. De plus, leur puissance d'expression limitée ne peut être palliée que par l'écriture de procédures à la charge de l'utilisateur, notamment lorsqu'il s'agit d'extraire de la base des informations implicites.

Au contraire, dans les systèmes de représentation de connaissances basés sur des objets, la connaissance est décrite de manière déclarative et se trouve séparée des mécanismes d'exploitation. Les structures de données (objets) effectivement manipulées par ce type de système sont entièrement dédiées à la représentation. Les mécanismes d'exploitation sont définis par des programmes écrits en n'importe quel langage chargés d'exploiter et de raisonner sur les bases de connaissances, dont ils sont effectivement séparés. Une logique de représentation est associée à ces systèmes; elle définit la sémantique des diverses entités du système (classes, instances, attributs) et permet de déduire des faits implicites des connaissances déjà établies, notamment par le mécanisme d'héritage qui repose sur la structuration en hiérarchies des classes d'une base.

A ce titre, le système TROPES, décrit ci-dessous, est un système de représentation de connaissance basé sur des objets.

3.2. Les entités TROPES

TROPES [Mari&90] est un système de représentation de connaissances par objets multi-points de vue qui repose sur l'approche **classes/instances** [Rech&89]. Une base de connaissances est partitionnée en **concepts** disjoints, qui décrivent chacun une famille d'entités du monde réel. Un concept d'une base de connaissances représente un ensemble d'**instances** de cette base. Les concepts étant disjoints, une instance n'appartient qu'à un seul concept. La notion de **point de vue** permet d'observer un objet selon des considérations différentes et d'obtenir des informations particulières à chaque point de vue.

Dans chaque point de vue d'un concept, les **classes** sont organisées en *une structure de spécialisation* ou *hiérarchie* dont la racine est une classe représentant toutes les instances du concept (Cf. Figure 1). Dans la hiérarchie, excepté pour les feuilles, une classe se décompose au niveau inférieur en classes disjointes (ses *sous-classes*) dont elle est la *sur-classe*. Une instance appartenant à une classe C est également instance de toutes les classes situées sur le chemin de la racine à C. La disjonction des classes assure que ce chemin est unique.

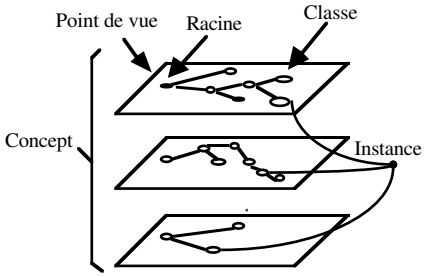

Figure 1 : Présentation des diverses entités TROPES

Un ensemble d'**attributs** est associé à chaque concept. Un attribut d'une classe représente une propriété, une relation ou un composant qui est une des

caractéristiques de la famille d'objets représentée par cette classe. Depuis chaque point de vue, une partie seulement des attributs du concept est visible. Un attribut peut prendre sa valeur dans un autre concept. Au niveau du concept, on regroupe des informations générales sur l'attribut et constantes dans tout le concept. Mais il est possible d'affiner la description d'un attribut au niveau d'une classe à l'aide de **descripteurs** ou **facettes**. Ces facettes—optionnelles—attachées à l'attribut permettent de définir pour la classe :

- le type de l'attribut (facette **dans**),
- la cardinalité ou nombre de valeurs de l'attribut (facette **card**),
- l'éclatement de l'attribut en plusieurs sous-attributs (facette **éclat**),
- le domaine des valeurs admissibles de l'attribut (facette **domaine**),
- des restrictions du domaine de valeurs de l'attribut (facette **sauf**),
- une valeur par défaut pour l'attribut (facette **defaut**) ,
- des conditions sur l'attribut (facette **cond**).

Une classe est décrite par une liste d'attributs (l'**intention**) et représente un ensemble d'instances (l'**extension**) qui sont les objets de la base créés sur le schéma de la classe, dont tout ou partie des attributs ont une valeur.

3.3. Les objets composites dans TROPES

Un objet **complexe** est un objet dans lequel au moins un des attributs a une valeur dont le type est l'intention d'une classe d'un concept. Si cet attribut intervient dans la composition de l'objet complexe (comme, par exemple, l'attribut "fuselage" de la classe "Avion de transport") et ne traduit pas simplement une propriété (comme, par exemple, l'attribut "nombre de passagers" de cette même classe), il est dit **attribut composant**. L'objet complexe devient alors **composite**. L'objet (instance) qui représente la valeur d'un attribut composant, le **composant**, est *dépendant de manière existentielle et exclusive* de l'objet composite qu'il compose. Le composant n'a été créé que pour l'objet composite et ne peut appartenir à aucun autre objet composite. Le type d'un attribut composant et sa cardinalité suffisent pour désigner la ou les valeurs de cet attribut. Lorsque l'attribut composant a plusieurs valeurs, on peut affiner son type par **éclatement**. L'éclatement a pour but de décrire séparément ou en les regroupant, les différents éléments simples et de même type dont l'ensemble indivisible forme le composant. Deux types d'éclatement sont possibles :

• un éclatement de la description de l'attribut, qui permet une décomposition et la vision immédiate de ses éléments.

Ex : l'attribut "atterrissage" décrivant l'enchaînement des procédures d'approche d'une piste d'atterrissage a pour description générale dans le concept "Vols" :

proc_atterrissage **liste de** contrôles

où "contrôles" `désigne une classe d'un autre concept. On peut éclater cet attribut composant dans la classe "Vol_sans_problèmes" en :

proc_atterrissage **liste de** échanges_tour | contrôles_instruments
card 5 | 10

• un éclatement de l'attribut lui-même, ce qui permet de distinguer chacune de ses parties et l'éclatement récursif du composant [Mari91].

Ex : l'attribut est éclaté en deux sous-attributs contrôles_internes et contrôles_externes :

proc_atterrissage
contrôle_externes **liste de** échanges_tour
card 5
contrôles_internes **liste de** contrôles_instruments
card 10

3.4. La classification dans TROPES

TROPES est doté d'un mécanisme de raisonnement puissant, la **classification**, qui consiste à déterminer, pour un objet de la base, la classe d'appartenance la plus spécialisée possible. La classification s'appuie sur l'**appariement** qui est la comparaison entre les valeurs des attributs de l'instance à classer et les contraintes imposées dans la définition de la classe par les facettes des attributs. Au regard de l'appartenance d'une instance à une classe, une classe est étiquetée comme **sûre** si l'instance vérifie toutes les contraintes de la classe, **impossible** si elle ne les vérifie pas, **possible** si les informations disponibles dans l'instance ne sont pas en contradiction avec les contraintes, mais ne sont pas suffisantes pour décider de l'appartenance à la classe.

Une fois l'appariement réalisé, la classification procède à une propagation d'étiquettes pour économiser de futurs appariements. La propagation d'étiquettes obéit aux règles suivantes :

- toute sur-classe d'une classe sûre est sûre,
- toute sous-classe d'une classe impossible est impossible,
- toute classe située au même niveau qu'une classe sûre est impossible.

L'idée majeure de la classification, lorsqu'il s'agit de classer un objet composite, est de charger le système de créer, d'après les descriptions des attributs composants, les instances qui deviendront les valeurs de ces attributs. Lorsqu'une classe candidate au rattachement d'une instance en cours de classification possède des attributs composants qui n'ont pas de valeur, le système se réfère à la description de ces attributs composants dans la définition de la classe candidate. Si ceux-ci sont décrits complètement par la donnée de leur type et de leur cardinalité, alors le système crée toutes les instances correspondant aux valeurs de ces attributs (création des composants), en les instanciant, dans leur concept respectif, dans les classes décrites par leur type.

Ce principe de classification d'objets composites par appariement de propriétés et instanciation des composants sera repris lors de l'exécution d'une tâche (Cf. §5).

4. La représentation des tâches dans le modèle TROPES

4.1. Le concept de "Tâche"

La tâche est considérée dans TROPES comme un objet à part entière. La puissance déclarative "naturelle" de ce système de représentation permet de rassembler dans un objet dédié à la description d'une tâche, les trois types d'informations relatives à son exécution et communes à toutes les tâches, à savoir : les entrées, le traitement et les sorties. La notion de tâche est représentée par un concept : le concept "Tâche". Celui-ci est un concept prédéfini et il est destiné, notamment, à représenter les tâches (les connaissances opératoires) disponibles et applicables sur les objets des divers autres concepts de la base.

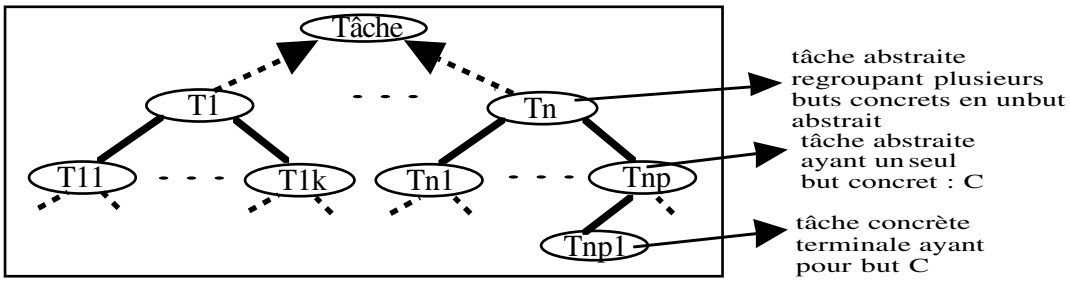

Figure 2 : Représentation du concept "Tâche". Dans une base de connaissances dédiée, par exemple, à l'analyse numérique, la classe T1 regroupe les opérations sur les entiers, Tn celles sur les matrices. Tnp est la tâche abstraite d'inversion de matrice et Tnp1, la tâche concrète d'inversion de matrice par la méthode de Choleski.

La classe racine de ce concept décrit l'ensemble potentiel de toutes les instances de tâches. Elle modélise tous les traitements possibles, c'est donc la tâche abstraite la plus générale. Une tâche abstraite est représentée par une sous-hiérarchie du concept qui caractérise l'aspect particulier de la tâche. On retrouve implicitement, dans les différents niveaux de la hiérarchie de classes, les différentes catégories de tâches considérées dans le modèle (Cf. Sec.2.1) : aux premiers niveaux, les tâches abstraites regroupant plusieurs buts, puis celles ayant un seul but, et, à la base, les tâches concrètes (Cf. Figure 2).

4.2. Description du concept de tâche

Après avoir introduit le concept "Tâche", il convient de décrire les principaux attributs attachés à ce concept.

• le **but**, qui est une chaîne de caractères correspondant au nom du but concret à atteindre. Un tel attribut n'a pas seulement un aspect convivial—le nom de la classe pourrait être à lui seul assez explicite—, mais le fait d'intégrer cette information dans la classe en tant qu'attribut permet de l'exploiter, comme, par exemple, lors d'une consultation de base de connaissances, dans le cadre d'un enseignement assisté par ordinateur, l'élève utilisateur pourra soit :

- obtenir l'ensemble des buts concrets disponibles en fonction d'un contexte particulier de la base,
- soit rechercher un but concret particulier.

• les **entrées**, qui sont en général spécifiques à une tâche particulière, mais peuvent occasionnellement être partagées par plusieurs tâches qui travaillent sur le même type de données (par exemple: inversion de matrice, transposée de matrice, ...). Une entrée est une donnée consommée par la tâche qui sera représentée par un nom et un type. A chaque entrée correspond, dans la description de la classe-tâche, une déclaration d'attribut.

• les **sorties** qui sont les données produites par la tâche. Chaque sortie sera représentée par un attribut.

• le **traitement** qui est une information dont le type est commun à toutes les tâches. Il doit exprimer la décomposition d'une tâche en sous-tâches, c'est-à-dire la séquence des tâches nécessaires à l'accomplissement de la tâche mère. Trois cas sont distingués :

1) la tâche est **abstraite**, il n'y a pas de traitement défini. La seule information disponible et minimale est son type : liste de tâches.

2) la tâche est **concrète non-terminale**, elle nécessite au moins un appel à une autre tâche. L'attribut **traitement** devra alors signaler les tâches à effectuer et l'ordre dans lequel elles vont être exécutées.

3) la tâche est **concrète terminale**, elle ne se décompose plus et fait appel à une procédure ou programme dont l'exécution n'est pas contrôlée par le système. Il n'y a pas de traitement.

La description du traitement est donc composée de zéro ou plusieurs tâches, c'est donc un ensemble de tâches. De plus, dans le cas où il n'est pas vide, il doit être ordonné suivant l'ordre d'exécution des sous-tâches. La notion d'ensemble ordonné, conduit alors à déclarer un attribut **traitement** comme étant une **liste de tâches.** C'est un attribut de nature **composant**.

Le troisième cas est unifié avec les deux autres en décidant qu'une tâche terminale correspond à une tâche dont l'attribut **traitement** est une liste **vide** de tâches. Toutefois, dans ce cas il est nécessaire d'indiquer qu'il y a exécution d'une **procédure**. Par ailleurs, il existe un concept "Procédure" qui représente l'interface entre le système TROPES et le monde extérieur. Elle fournit le mode d'emploi (langage, site d'exécution, chemin d'accès, transformations des données,…) de l'exécution d'un programme externe au système. Une tâche terminale possède alors un attribut **exécution** de **type procédure** déclaré dans le concept "Tâche".

4.3. Création de la structure de classes

La collection d'attributs du concept "Tâche" se divise en trois catégories majeures : entrées, traitement, sorties. L'ordre d'apparition de ces attributs, dans la définition des classes de tâches, a un lien direct avec la hiérarchie de spécialisation de tâches. En effet, aux premiers niveaux, les tâches abstraites se spécialisent en fonction du but et des entrées, le traitement restant général. Plus bas dans la hiérarchie, le traitement est fixé graduellement (par éclatement et affinement d'attributs). Finalement, les niveaux les plus bas sont structurés en fonction des sorties.

5. L'exécution de la tâche

Chaque instance du concept "Tâche" représente une exécution réussie de tâche. La notion de réussite ou d'échec d'exécution d'une tâche en fonction d'un contexte d'utilisation s'exprime, dans la hiérarchie de classes, en terme d'appartenance ou non à la classe qui représente cette tâche, d'une instance dont les attributs entrées représentent le même contexte d'utilisation. Dans cette optique, le processus d'exécution commence par une instanciation de la classe correspondant à la tâche à résoudre.

Instancier une classe, éventuellement à partir de la donnée de certaines valeurs d'attributs, consiste à créer une instance de cette classe d'après sa description et celle de ses sur-classes (héritage), à s'assurer alors que les valeurs fournies ne sont pas en contradiction avec les contraintes associées à la définition de la classe et de ses sur-classes (auquel cas l'instanciation échoue), puis, à inférer, si possible, toutes les valeurs manquantes de l'instance créée. Dans le cadre de l'exécution d'une tâche, cela revient à créer une instance avec les valeurs d'entrées de la tâche et à déterminer la valeur de ses attributs traitement et sorties.

Le processus d'exécution d'une tâche comporte une première étape d'instanciation de la tâche à résoudre, une deuxième étape de classification (ou descente) dans la hiérarchie de classe en vue d'obtenir le traitement adéquat et finalement une troisième étape d'instanciation des attributs composants décrivant le traitement de la tâche concrète choisie. L'instanciation de ces attributs composants peut déclencher récursivement ce processus. L'exécution d'une tâche est , à présent, illustré sur un exemple.

5.1. Exécution de l'objet composite tâche

Avant de généraliser au cas de la tâche abstraite, il convient de décrire tout d'abord l'instanciation d'une tâche concrète.

• Instanciation de tâche concrète (Figure 3) :

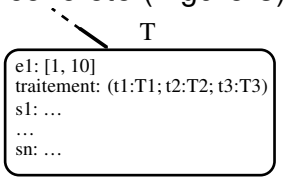

Figure 3 : Description d'une classe de tâche. La classe T est une tâche concrète à une seule entrée entière e1, définie à ce niveau, dont la valeur se trouve dans [1,10]. Le traitement est l'éclatement de la liste décrivant la séquence des tâches t1, t2 et t3 de type respectif T1, T2 et T3.

Pour une tâche concrète, le traitement est unique. Sur l'exemple, il constitue une séquence de tâches, chacune définie par la classe à laquelle devra appartenir l'instance correspondante de tâche. Ici, comme pour l'instanciation d'un objet composite, l'instanciation de la classe T crée une instance de cette classe avec les valeurs d'entrées fournies si les contraintes des sur-classes de T sur les attributs hérités et la contrainte associée à la seule entrée e1 (de type entier et de valeur dans l'intervalle [1,10]) sont vérifiées. Si les valeurs ne sont pas valides,

l'instanciation échoue. Sinon, la valeur de l'attribut traitement est inférée par les instanciations successives (selon la séquence) des classes représentant les tâches T1, T2 et T3.

L'instanciation de T échoue si l'une de ces instanciations échoue ou si les sorties[2] ne sont pas valides; sinon l'instanciation de T réussit.

• Exécution de tâche abstraite :

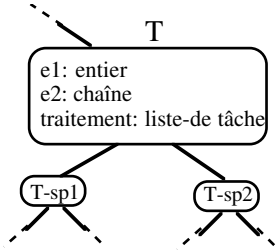

Figure 4 : T est une tâche abstraite dont T-sp1 et T-sp2 sont des spécialisations.

L'exécution d'une tâche abstraite débute par une instanciation de la classe correspondante T (Cf. Figure 4), qui se distingue du cas précédent par l'inférence de la valeur du traitement. En effet, la cardinalité n'étant pas fixée pour cet attribut, l'instanciation des sous-tâches ne peut, a priori, être lancée.

Dans ce cas, le système utilise une connaissance a posteriori sur l'exécution de tâches : lorsque l'exécution d'une tâche abstraite réussie, c'est que l'une des tâches concrètes dont elle recouvre le cas a pu être exécutée et donc lorsqu'une de ses spécialisations (concrètes ou abstraites) a réussi.

Ainsi, l'instanciation de T est déléguée à l'instanciation de T-sp1 ou T-sp2. C'est toujours le principe de la classification d'objet composite (descente de l'instance créée de l'objet composite tâche) qui est appliqué ici à partir de la classe T. L'instanciation de T réussie si la descente s'est effectuée jusqu'à une tâche concrète dont l'instanciation a réussi. Sinon, l'instanciation de T échoue.

5.2. Un exemple d'exécution de tâche

L'exemple choisi (Cf. Figure 5) est celui de l'exécution d'une tâche dont le but est d'inverser une matrice. Il est demandé au système d'inverser une matrice correspondant à un objet du concept "Matrice" de la base.

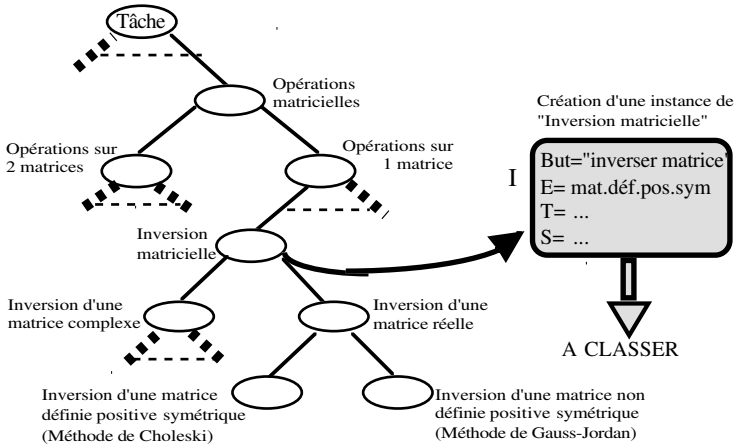

Figure 5 : Détermination et instanciation de la tâche abstraite "inversion matricielle". L'instance créée a pour entrée une matrice définie positive et symétrique. La classification (descente dans la hiérarchie) permettra de valuer le traitement et la sortie.

Le système procède d'abord par la sélection sur le but. Ici, le but à atteindre est "inverser une matrice". De la hiérarchie du concept "Tâche", il ressort que la tâche abstraite "Inversion de matrice" est la tâche la plus générale, dont le but

[2] La description du flot de données, liaison entre les entrées/sorties de la tâche appelante et les entrées/sorties de ses sous-tâches, n'est pas abordée dans cet article.

concret—l'attribut **but** est défini et a une valeur unique—correspond à la requête. Une fois, cette classe de tâche abstraite à but concret désignée, le système se charge de créer une instance de cette classe dont les attributs entrées contiennent les données à traiter.

Dès lors, le système doit trouver une classe de tâche plus spécifique en fonction des entrées, puis une classe dans laquelle l'attribut **traitement** est suffisament défini pour que les instanciations puissent être lancées.

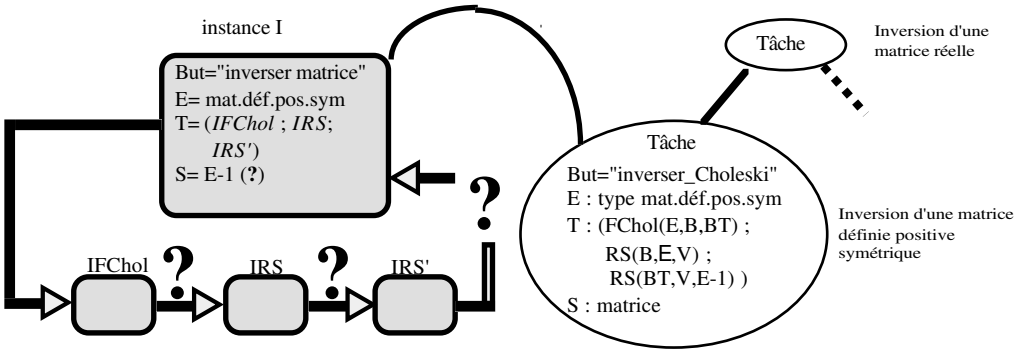

Figure 6 : La classification de l'instance selon ses entrées est terminée. Dans la classe (inversion par la méthode de Choleski), l'attribut **traitement** est défini comme la liste de tâches : • FChol(E, B, B^T) : Factorisation de Choleski de la Matrice E = BB^T

• RS(B, E, V)) : Résolution des n systèmes $Bv_j = E_j$ pour $1 \leq j \leq n$ où E_j est un vecteur de la base canonique

• RS(B^T, V,E^{-1}) : Résolution des n systèmes $B^T u_j = v_j$ pour $1 \leq j \leq n$ où u_j est le jème vecteur colonne de E^{-1}.

Le système tente alors les 3 instanciations dont la réussite dirige l'exécution de la tâche.

La matrice de l'exemple étant réelle, symétrique et définie positive l'instance descend jusqu'à la classe de tâche concrète "inversion d'une matrice définie positive symétrique ou méthode de Choleski). Dans cette classe (Cf. Figure 6), l'attribut **traitement** est entièrement défini comme la séquence de trois tâches dont les classes sont fournies. Le système recommence alors le processus d'exécution en créant une instance de tâche de la première classe désignée dans la séquence, puis, si l'exécution de cette tâche réussie, une instance de la deuxième classe de la séquence et, enfin, en cas de succès, une instance de la troisième classe de la séquence.

Trois exécutions de tâches seront donc lancées (trois instanciations à réaliser). De leur validation dépend la réussite de l'exécution de la tâche "inverser matrice" pour la matrice réelle définie positive symétrique donnée à traiter.

6. Conclusion

Le modèle de tâches présenté est entièrement intégré et exprimé dans le modèle de représentation de connaissances par des objets TROPES. La représentation de l'exécution d'une tâche est elle-même construite entièrement par des mécanismes d'inférence de TROPES, l'instanciation et la classification. Le résultat d'une telle exécution est une trace représentée par un objet composite.

L'exploitation de la trace par le système, par exemple avec des techniques d'appariement d'objets complexes, permet de raisonner sur la trace elle-même, donc sur le raisonnement lui-même. Ceci offre d'intéressantes perspectives pour le développement des capacités explicatives, de reconnaissance d'intention et d'apprentissage du système.

Dans le même ordre d'idée, il est envisageable d'exprimer les mécanismes d'inférence de TROPES, tels que la classification ou l'instanciation, etc., en termes de tâches. Ceci permettrait de les paramétriser et d'avoir une trace de leur exécution. De plus, la gestion de points de vue multiples dans TROPES devrait permettre d'exploiter une trace d'exécution d'après diverses considérations.

7. Bibliographie

[Bobr&81] D. G. Bobrow, M. Stefik, **The Loops Manual**, Technical Report KB-VLSI-81-13, Knowledge Systems Area, Xerox Palo Alto Research Centers, 1981.

[Chan86] B. Chandrasekaran, **Generic Tasks in Knowledge-Based Reasoning : High-Level Building Blocks for Expert System Design**, IEEE Expert, pp 23-30, Fall 1986.

[Clan85] W. J. Clancey, **Heuristic Classification**, AI Journal, N°4, 1985.

[Davi&90] J.M. David, J.P. Krivine, **Structuration du raisonnement et explications, r**evue d'intelligence artificielle. Volume 4, pp 77-88, 1990.

[Gold&83] A. Goldberg, D. Robson, **Small talk-80: The Language and its implementation**, Addison-Wesley, Reading, Mass., 1983.

[Hasl&84] Hasling D.W., Clancey W.J., G. Rennes, **Strategic explanations for a diagnostic consultation system**, International Journal of Man-Machine Studies, 20, pp 3-19, 1984.

[Mari&90] O. Mariño, F. Rechenmann, P. Uvietta, **Multiple Perspectives and Classification Mechanism in Object-Oriented Representation**, ECAI'90 Proceedings, Stockholm, Londres, Pitman Publishing, août 1990.

[Mari91] O. Mariño, **Classification d'objets composites dans un système de représentation de connaissances multi-points de vue**, 8ème congrès RFIA, Lyon, 25-29 novembre 1991.

[Naul&91] Nault J.P., Schmeltzer O., **La reconnaissance d'intentions par appariement d'objets**, Rapport Année Spéciale IA, ENSIMAG, INPG, Grenoble, juin 1991.

[Pate90] P.F. Patel-Schneider, **Practical, Object-based Knowledge Representation For Knowledge-based Systems**, Information Systems, Vol. 15, N°1, pp 9-19, 1990.

[Ponc&91] T. Poncabare, F. Rechenmann, **SCAI : un environnement de développement de systèmes à base de connaissances en calcul scientifique et technique**, Convention IA 91, Hermes, Paris, 1991.

[Rech&89] F. Rechenmann, P. Fontanille, P. Uvietta, **Shirka : manuel d'utilisation**, Doc. interne, INRIA - Laboratoire ARTEMIS / IMAG, Grenoble, 1989.

[Smit&84] R.G. Smith, G.E. Lafue, E. Schoen, S.C. Vestal, **DeclarativeTask Description as a User-Interface Structuring Mechanism**, IEEE Computer, September 1984.

[Stee90] L. Steels, **components of expertise**, AI Magazine, Summer 1990.

[Uvie&91] P. Uvietta, J. Willamowski, **Modélisation de connaissances en biologie moléculaire**, 8ème congrès RFIA, Lyon, 25-29 novembre 1991.