Intégration de contraintes, d'objets composites et de tâches dans un modèle de représentation de connaissances par objets

Integrating constraints, composite objects and tasks in an object-based knowledge representation system

Jérôme GENSEL   Pierre GIRARD   Olivier SCHMELTZER

INRIA - Rhône-Alpes – IMAG/LIFIA
46, avenue Félix Viallet
F-38031 GRENOBLE CEDEX 1
Tél.: (33) 76.57.45.90.
Fax: (33) 76.57.46.95.
e-mail: Jerome.Gensel@imag.fr

Résumé :
TROPES est un système de représentation de connaissances par objets qui incorpore trois caractéristiques importantes : les contraintes, les objets composites et les tâches. Les contraintes procurent un moyen déclaratif de définir et de maintenir des relations au sein d'objets et entre objets. La relation de composition tire profit de celles-ci pour la description d'un mécanisme de partage de valeurs entre un objet composite et ses composants. Le modèle de tâches utilise à la fois les objets composites pour la modélisation des tâches et les mécanismes de classification et d'instanciation pour leur exécution; il repose également sur des contraintes pour la propagation des entrées et des sorties d'une tâche à l'autre.

Abstract :
TROPES is an object-based knowledge representation system, offering three important intertwined features: constraints, composite objects and tasks. Constraints provide a declarative means to define and maintain relations inside and between objects. The part-whole relation profits by constraints for sharing attribute values between a composite object and its components. The task model benefits from composite objects for task modeling and from classification and instantiation mechanisms for task execution; it also uses constraints to propagate input and output data from one task to another.

Mots-clés :
Représentation de connaissances, objets, contraintes, objets composites, tâches.

Keywords :
Knowledge representation, objects, constraints, composite objects, tasks.

1 Introduction

Les systèmes de représentation de connaissances sont reconnus pour leur capacité à modéliser à la fois les entités d'un domaine et les mécanismes destinés à les exploiter. Ces systèmes se fondent sur des modèles déclaratifs d'expression de la connaissance qui permettent d'exprimer celle-ci sans présager de son utilisation.

Les systèmes de représentation par objets (SRPO), issus des "frames" [Fikes et al.85], en sont un bon exemple dans lesquels l'*objet* constitue l'unité de description de la connaissance. Cette notion d'objet est à distinguer de celle utilisée dans les langages de programmation orientée objet (LPOO) puisque, par opposition à ceux-ci, les SRPO séparent clairement la connaissance des mécanismes d'exploitation chargés de l'étendre [PS90]. Une logique de représentation définit alors la sémantique des diverses entités manipulées (classe, instance, attribut) et permet de déduire des faits implicites des connaissances déjà établies.

A ce titre, TROPES (dont une brève présentation est faite dans la section 2), est un SRPO conçu pour supporter un raisonnement classificatoire similaire à celui exploité par SHIRKA [Rechenmann et al.90], son prédécesseur, et par les langages terminologiques [McGregor et al.92, Brachman et al.85].

L'intégration de contraintes (section 3) non seulement accroît la puissance descriptive du modèle, mais permet également, lors de l'introduction des objets composites (section 4), de traiter naturellement la propagation de valeurs entre un objet composite et ses composants. Enfin, l'expression de la notion de tâche (section 5) en termes d'objet composite et de contrainte illustre les capacités de représentation du modèle TROPES ainsi étendu.

2 Présentation du modèle Tropes

2.1 Entités de base

TROPES [Marino et al.90] est un système de représentation de connaissances par objets multi-points de vue basé sur une approche classe/instance. En TROPES, une *base de connaissances* est partitionnée en *concepts* indépendants qui modélisent les différentes familles d'individus. Plusieurs *points de vue* peuvent être associés à un concept, chacun d'eux permettant d'en donner une interprétation particulière. A un point de vue correspond un ensemble de *classes* organisées en hiérarchie par la relation de *spécialisation* qui traduit l'inclusion de l'ensemble des *instances* d'une classe – l'*extension* de la classe – dans celui d'une sur-classe. Dans une telle hiérarchie, excepté pour la *racine*, une classe a une sur-classe directe unique (la hiérarchie est donc un arbre). De plus, les extensions des sous-classes directes d'une même classe sont disjointes (la multi-instanciation est impossible dans un même point de vue). Par conséquent, une instance est attachée à une et une seule classe dans chaque point de vue par le lien 'est-un' et appartient à chaque classe située sur le chemin de cette classe à la racine (Figure 1).

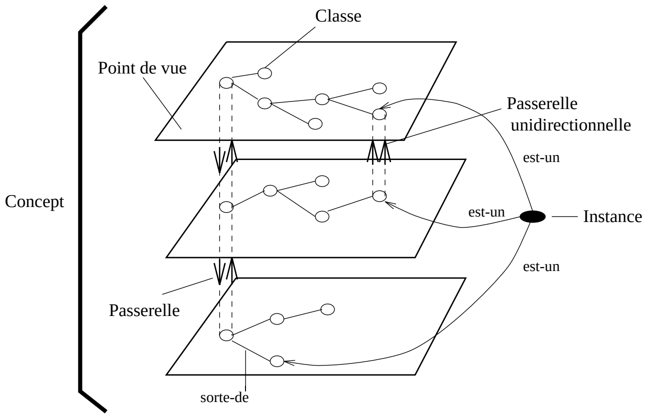

FIG. 1 - : Les principales entités du système TROPES

Les points de vue sur un concept représentant des catégorisations (organisations en hiérarchie de classes) différentes du même ensemble d'instances, il peut être souhaitable de souligner les relations existant entre classes de points de vue distincts. Ceci est réalisé par la définition de *passerelles*. Dans sa forme la plus simple, une passerelle impose l'inclusion d'une classe source dans une classe destination.

Une classe donne une description – l'*intension* de la classe – qui est constituée d'une collection de descriptions d'*attributs*, celles définies dans cette classe et celles définies dans ses sur-classes (obtenues par héritage). Les valeurs des attributs d'une instance ne peuvent violer l'intension de ses classes d'appartenance (i.e. les contraintes définies pour ces classes). Ainsi, la construction incrémentale des descriptions de classe dans la hiérarchie assure l'inclusion de l'extension de chaque classe dans celle de sa sur-classe.

Les attributs sont définis dans et pour un concept, chacun d'eux permettant d'exprimer soit une propriété, soit un lien 'partie-de' (cf section 4), soit un lien relationnel, d'une instance de ce concept. Un attribut a une définition pour tout le concept qui en décrit les caractéristiques générales (mono ou multi-valué, son type initial, ses méthodes de calcul,...), et une définition pour chacune des classes où il est affiné. L'affinement d'un attribut dans une classe se fait à l'aide de facettes optionnelles et permet de préciser pour cette classe :

- son nouveau type, qui est un sous-type de celui défini précédemment (facette 'dans'),
- le domaine de ses valeurs possibles (facette 'domaine'),
- le domaine de ses valeurs impossibles (facette 'sauf'),
- sa cardinalité, quand l'attribut est du type construit liste, ou ensemble (facette 'card'),
- les éléments d'un attribut de type liste, ou ensemble, par éclatement de cet attribut en sous-attributs (facette 'éclat').

Les entités présentées offrent donc à l'utilisateur un moyen de décrire les bases de connaissances. L'exploitation de ces bases (i.e. définition et complétion des instances) est réalisée par divers mécanismes présentés dans la section suivante.

2.2 Mécanismes de gestion d'instance

Les opérations décrites ici concernent uniquement la manipulation de la notion d'instance. Toutefois, des mécanismes de validation, de structuration et d'acquisition des connaissances (manipulation de la notion de classe), s'appuyant sur la notion de type, sont également proposés dans TROPES; ils sont décrits dans [Capponi93].

2.2.1 Création et rattachement d'instance

L'instanciation est le mécanisme qui permet de créer une instance à partir de la spécification de son concept, de la classe à laquelle elle doit être rattachée dans chaque point de vue (la racine par défaut) et de paires (attribut, valeur).

L'instanciation échoue dès qu'une valeur d'attribut viole la description d'une des classes de rattachement.

L'instance créée avec succès peut se révéler incomplète si la valeur inconnue '?' a été fournie dans une paire (attribut, valeur), ou si la définition d'une des classes de rattachement comporte un attribut non spécifié. Des calculs de valeurs peuvent alors être déclenchés pour ces attributs inconnus.

2.2.2 Calcul de la valeur d'un attribut

En TROPES, il existe deux moyens d'obtention de la valeur d'un attribut d'une instance : par propagation des contraintes définies dans les classes d'appartenance de l'instance à compléter, ou par appel à l'ensemble des tâches associé à l'attribut dans le concept.

Dans le premier cas, chaque contrainte d'une classe est activée lorsque l'instance y est rattachée. Si l'information disponible dans l'instance est suffisante, un attribut indéterminé mais contraint pourra être évalué (cf section 3).

Dans le second cas, le calcul de la valeur de l'attribut est lancé lorsque la valeur est requise par un utilisateur ou par un des mécanismes d'exploitation de TROPES. Ce moyen de calcul est considéré comme une connaissance globale du concept, indépendante d'un rattachement particulier de l'instance à une classe. Cette connaissance est contenue dans un ensemble de tâches organisé en hiérarchie; elle est fournie par la définition générale de l'attribut pour le concept. Le choix d'une méthode de calcul (i.e. d'une tâche) pour l'attribut se fait donc en fonction des informations contenues dans l'instance par le mécanisme d'exécution de tâche décrit section 5.

Réunir et organiser au niveau du concept l'information opératoire concernant les attributs de ce concept est justifié dans [Rechenmann92] sous le terme de *détachement procédural*. En TROPES, l'attachement procédural n'est possible que sous la forme d'une contrainte de classe.

2.2.3 Classification d'instance

La classification d'instance est un mécanisme qui consiste à confronter la connaissance acquise dans l'instance aux différentes hiérarchies (points de vue) de classes du concept. L'objectif est de déterminer pour chaque point de vue, l'ensemble des classes d'appartenance de l'instance (en fait, les plus spécialisées sont suffisantes en vertu de la sémantique de la spécialisation). Le principe de l'algorithme repose sur un marquage des classes comme "Possible", "Impossible" ou "Sûre", d'après la confrontation du contenu de l'instance à la description de la classe candidate, durant un parcours descendant dans chacun des points de vue (pour plus de détails, cf [Marino et al.90, Marino91]). Chacune de ces marques reflète le degré d'adéquation entre les informations portées par l'instance et celles décrites par la classe.

Ce mécanisme s'avère être un support puissant pour enrichir la connaissance sur une instance. En effet, si dans son mode le plus simple – le mode "manuel" – il ne fait que proposer les résultats du marquage, laissant à l'utilisateur toute initiative de rattachement et de déclenchement de calcul d'attribut, dans son mode le plus prospectif – le mode "automatique"– il attache, lors de la descente, l'instance à la première classe jugée "Sûre" et déclenche en fonction des besoins les calculs d'attributs nécessaires (cas où il n'y a pas de classe "Sûre" mais au moins une classe "Possible").

Après avoir donné un rapide aperçu des éléments de structuration de la connaissance et des mécanismes permettant son exploitation, l'accent va être mis sur trois extensions de Tropes – les contraintes, les objets composites et les tâches – et sur les liens qui les associent en un ensemble compact de fonctionnalités.

3 Intégration de contraintes

3.1 Motivations

Disposer de méthodes peut sembler suffisant pour calculer la valeur d'un attribut dans un modèle de représentation de connaissances. Cependant, lorsque l'on souhaite exprimer une relation entre des attributs, on est contraint d'associer à chacun d'eux une méthode du calcul de sa valeur en fonction des valeurs des autres attributs. La relation n'est donc pas donnée explicitement mais est exprimée implicitement à travers un ensemble diffus de méthodes.

Recourir à l'emploi d'un langage de contraintes pour représenter et maintenir de telles relations est une solution communément admise aujourd'hui dans divers courants de l'Intelligence Artificielle [Kumar92]. Ces langages permettent d'énoncer un problème de satisfaction de contraintes ou CSP par la donnée d'un ensemble C=$\{C_1, \ldots, C_n\}$ de n contraintes sur un ensemble V = $\{x_1, \ldots, x_m\}$ de m variables (appelées variables contraintes), possédant chacune un domaine D_{x_i} de valeurs dont le type varie selon le langage (entiers, booléens, réels, objets, ...). Chaque contrainte $C_i(x_{i1}, \ldots, x_{im})$ de ce CSP décrit une relation à maintenir sur un sous-ensemble $V_{C_i} = \{x_{i1}, \ldots, x_{im}\}$ de V [Montanari74]. Backtrack, propagation de contraintes, heuristiques d'ordonnancement des variables à instancier et des valeurs à choisir sont les techniques les plus couramment utilisées par les résolveurs de ces langages dans la phase de satisfaction du CSP.

L'objectif de l'intégration de contraintes à Tropes est de permettre l'expression déclarative de contraintes et le maintien de relations entre plusieurs attributs, sans s'éloigner du formalisme et des entités du modèle.

3.2 Représentation des contraintes

3.2.1 Les classes de contraintes

Dans un modèle de représentation de connaissances au moyen d'objets, il semble naturel de décrire les contraintes également par des objets. A l'instar des méthodes, la classe est l'unité de description d'une contrainte. L'instanciation d'une telle classe correspond à la pose effective de la contrainte. Une instance de classe de contrainte constitue donc une trace de la présence de cette contrainte sur des attributs. Grâce à ce mode de représentation, la définition, l'activation et la modification des contraintes se font à l'aide des primitives classiques de manipulation offertes par Tropes.

Intégrant les contraintes en tant qu'objets, Tropes est à rapprocher davantage de l'implémentation du système Prose [Berlandier92] en Le-Lisp V16 [Ilog s.a.92a] où contraintes, variables et relations sont modélisées par des classes de Telos, le langage de programmation par objets intégré, que de systèmes comme Thinglab [Borning81] ou Kaleidoscope [FB90] dans lesquels les contraintes sont exprimées par des règles à l'intérieur des objets contraints.

Trois types de contraintes modélisées par des classes (contraintes de classe, contraintes entre instances et contraintes de relations) sont disponibles dans Tropes. Les contraintes de classes – les seules présentées ici (cf [Gensel93] pour plus de détails) – interviennent également dans l'expression du partage de valeurs entre un objet composite et ses composants et du flot de données entre tâches. Elles constituent donc un outil précieux pour l'utilisateur et pour le modèle lui-même.

3.2.2 Les contraintes de classe

Une relation entre des attributs d'une même classe, est représentée en TROPES par une contrainte de classe. De fait, chaque instance de la classe y est alors soumise. Dès que l'information est suffisante, une contrainte de classe est capable de déclencher la méthode d'inférence de la valeur d'un attribut contraint. Ces méthodes définies par la contrainte peuvent donc être vues comme des méthodes associées à la classe. Ce n'est pas en contradiction avec les raisons du détachement procédural, si l'on considère que la contrainte exprime une propriété de cette classe (cf Figure 2).

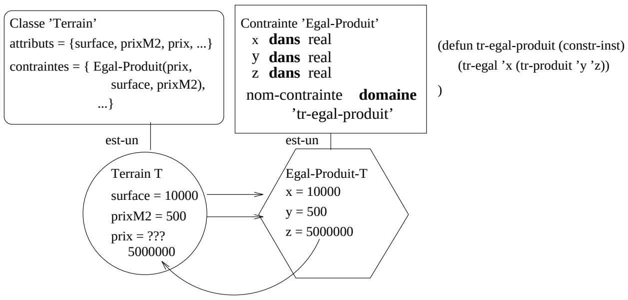

FIG. 2 - : Une contrainte lie les attributs 'surface', 'prixM2' et 'prix' de la classe "Terrain". Dès que deux de ces attributs sont connus (ici, dans l'instance 'Terrain T', 'surface' et 'prixM2'), la méthode d'inférence dédiée au troisième ('prix') est activée permettant de déduire sa valeur. Les trois valeurs satisfont alors la contrainte. A chaque instance de "Terrain" créée, correspond une instance de contrainte 'Egal-Produit' qui représente la contrainte, les attributs contraints et leur domaine ou valeur. Les fonctions 'tr-egal' et 'tr-produit' sont des fonctions fournies par l'interface TROPES-PECOS

Une instance en particulier ne peut être libérée d'une contrainte de classe qui porte sur elle. Ces contraintes sont imposées par la classe. En revanche, une classe peut être libérée d'une contrainte à tout moment, ce qui annule pour toutes les instances, l'inférence de cette contrainte et celles qu'elle a permises.

Une contrainte de classe placée sur la destination d'une passerelle est également reportée sur la source (en raison de l'inclusion requise des extensions).

Les contraintes de classe sont héritées le long du lien de spécialisation 'sorte-de' et de la classe destination vers la classe source d'une passerelle. Enfin, les contraintes étant des classes, des contraintes sur les contraintes pourraient être définies et les types des attributs contraints pourraient être affinés dans des sous-classes de contraintes, mais ces éventualités n'ont pas encore été approfondies...

3.3 Satisfaction des contraintes

Si l'expression des contraintes est entièrement réalisée par TROPES, leur satisfaction est déléguée à PECOS [Ilog s.a.92b], la librairie de programmation par contraintes du langage LE-LISP.

3.3.1 Les apports de Pecos

PECOS permet de définir des variables contraintes de tout type auxquelles un domaine de valeurs (énuméré ou intervalle) peut être associé. Un CSP sur ces variables contraintes peut alors être énoncé à l'aide de contraintes prédéfinies (d'égalité, arithmétiques, ...) ou décrites par l'utilisateur grâce à des *schémas* de définition. La définition d'une contrainte est basée sur un mécanisme de démons qui activés (par instanciation ou réduction du domaine d'une variable contrainte) déclenchent les règles de propagation qui réduiront les domaines des variables attenantes.

Des primitives de programmation en mode non déterministe sont disponibles qui, basées sur un backtrack, permettent d'énumerer des domaines dans la recherche de solutions d'un CSP ou de revenir à des points de choix explicites.

Afin que TROPES puisse bénéficier de la puissance de résolution de PECOS et de son module de définition de contraintes, une interface entre les deux systèmes a été conçue.

3.3.2 L'interface Tropes/Pecos

L'interface TROPES/PECOS offre :

- un éventail de contraintes de base écrites à partir de fonctions PECOS et chargées de la gestion entre attributs contraints de TROPES et variables contraintes PECOS,
- une adaptation des principes d'utilisation d'une base de connaissances aux variables contraintes de PECOS (modification de valeur permise, gestion de variables contraintes composite lorsque des attributs

contraints de TROPES ont un domaine résultant d'unions de domaines, ...),
- report dans TROPES des inférences, réductions de domaines ou erreurs réalisées par PECOS.

Ce couplage permet à TROPES d'exploiter la propagation locale des valeurs sur laquelle repose la satisfaction des contraintes dans PECOS: dès que le domaine d'un attribut contraint est réduit à un singleton, cette valeur peut être associée de droit à l'attribut en tant que seule valeur possible. L'intégration de contraintes offre donc un moyen d'inférence supplémentaire dans TROPES. Toutefois, il serait erroné de voir les contraintes comme une fonctionnalité ajoutée à TROPES dans le seul but d'en augmenter les capacités de représentation et de résolution de problèmes. En effet, celles-ci constituent un mécanisme exploité par le modèle lui-même dans la gestion des objets composites et des tâches comme le montrent les sections suivantes.

4 Les objets composites

L'introduction d'objets composites dans le modèle permet d'y ajouter une relation supplémentaire dont la sémantique diffère de celle de l'attribution. De plus, la relation qui lie un tout et ses parties s'enrichit d'un mécanisme de partage de valeurs qui lui est propre.

4.1 La relation partie-tout

Après avoir choisi une sémantique à la relation de composition, son comportement vis-à-vis des liens entre un composé et les composants est décrit.

4.1.1 La sémantique de la relation de composition

Winston *et al.* [Winston et al.87] ont spécifié six relations différentes et les ont classées selon trois critères : la *fonctionnalité* - les parties possèdent un rôle fonctionnel par rapport au tout; l'*identité* - les parties sont similaires entre elles et au tout; la *séparabilité* - les parties peuvent être séparées du tout auquel elles appartiennent. Ces six relations sont les suivantes : composant-objet entier (anse-tasse), membre-ensemble (arbre-forêt), portion-masse (part de tarte-tarte), matière-objet (acier-vélo), caractéristique-activité (payer-faire des courses).

Dans TROPES, la relation utilisée est celle liant le composant à l'objet entier (même si a priori, aucune vérification ne peut être faite par le système), car elle possède les propriétés de fonctionnalité et de séparabilité. De plus, on la rencontre très souvent et elle autorise la décomposition récursive des objets; ce dernier point apporte un mécanisme d'inférence supplémentaire, la transitivité de cette relation (i.e. si un objet composite est composé d'un autre objet composite, les composants de ce dernier sont aussi ceux du premier).

4.1.2 Le comportement des objets composites

Quatre types de comportement existent suivant la dépendance et le partage [Kim et al.89]. Un composant est dépendant de son composé s'il ne peut pas exister sans lui. Dans ce cas, la destruction d'un objet composite implique celle de ses composants. Un composant est partagé si plusieurs objets composites l'utilisent comme composant.

Dans TROPES, le devoir de dépendance est restreint à la destruction des composants après celle de l'objet composite; néanmoins, il peut exister des composants libres qui seront éventuellement incorporés à des objets composites.

Par contre, il découle du choix de la relation (composant-objet entier) qu'un composant ne peut être partagé par plusieurs objets composites, ce qui ne peut se produire que pour des entités logiques. C'est pourquoi, il n'est pas possible de modifier un composant sans passer par l'objet composite auquel il appartient.

4.1.3 Les objets composites dans Tropes

Les objets composites sont introduits très naturellement dans TROPES en créant deux types d'attributs : les attributs classiques font office de propriété et caractérisent les entités (relation d'attribution) tandis que les attributs composites pointent sur les composants. Ces deux types d'attributs réagissent cependant de la même manière vis-à-vis de l'héritage.

Puisque la relation de composition est modélisée grâce aux attributs composites, un même objet peut être décomposé autant de fois qu'il y a de points de vue. Par exemple, une voiture peut être décomposée, selon un point de vue mécanique, en le moteur, le système de freinage, la transmission, etc, et, selon un point de vue physique, en ses portes, sa carrosserie, etc.

Enfin, un attribut composite multi-valué peut être éclaté en plusieurs autres attributs à l'instar des autres attributs.

4.2 Le partage de propriétés

L'ajout d'une nouvelle relation est important car cela clarifie la sémantique des liens qui existent entre les entités manipulées. Un autre intérêt est d'adjoindre à cette relation des mécanismes d'inférences spécifiques.

4.2.1 La problématique

Supposons qu'une classe *Voiture* possède un attribut *Couleur* et un composant *Porte* qui, lui aussi, a un attribut *Couleur*. La couleur de la voiture et celle de la porte peuvent être liées de différentes manières selon que l'une est tirée de l'autre ou réciproquement. L'héritage ainsi défini est appelé *co-héritage*. Un attribut d'un objet composite hérite la valeur de ses composants ou le contraire selon ce qu'on veut exprimer. Cela devient plus complexe quand on veut faire des opérations sur les valeurs des attributs, comme par exemple exprimer que le poids d'un vélo est égal à la somme des poids de son cadre, sa selle, ses roues, etc.

La propagation d'une valeur de l'objet composite à ses composants ne pose pas de problème de compatibilité puisque seule une valeur est propagée (car un composant ne peut pas appartenir à plusieurs objets composites). Dans l'autre sens, par contre, si les composants ont des valeurs différentes, laquelle propager à l'objet composite? Nous adoptons ici une propagation *paresseuse* en ne propageant une valeur que si les composants ont la même. De plus, le partage de valeurs est *dynamique* en ce sens que le mécanisme de propagation n'est activé qu'à la demande d'une valeur d'un attribut bénéficiant de facettes de co-héritage. Enfin, il est possible d'exprimer des inférences plus compliquées en spécifiant un opérateur à appliquer sur les composants (par exemple la somme).

4.2.2 L'implémentation du co-héritage

Deux nouvelles facettes ont été ajoutées aux attributs classiques : la première dit si la valeur de l'attribut peut être inférée d'un composant ou d'un composite – cette facette appelée *co-héritage* peut prendre trois valeurs, "composite" si l'attribut tire sa valeur du composite, "composant" s'il la prend chez ses composants et "indifférent" si les deux sont possibles; la seconde (*co-opérateur*) spécifie l'opérateur à appliquer pour déterminer cette valeur : "égalité" si les valeurs doivent être égales, "moyenne" si la valeur de l'attribut dans l'objet composite est la moyenne de celles chez les composants, "somme" si c'est la somme, "max" ou "min" si c'est la plus grande ou la plus petite (Fig. 3). Cette information sur le calcul de la valeur se met exclusivement au niveau de l'objet composite. Ces opérateurs augmentent la déclarativité de la base de connaissances en apportant une information compréhensible dans les objets eux-mêmes; il est cependant également possible de définir d'autres opérateurs avec une sémantique particulière.

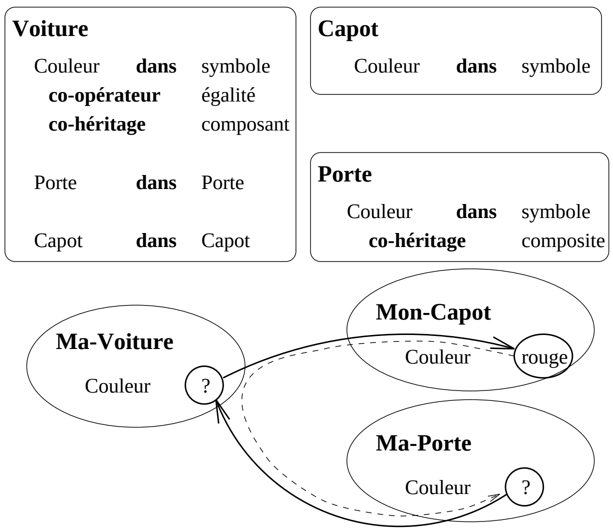

FIG. 3 - : La propagation de valeurs d'attributs. La description des classes montre que la valeur de l'attribut *couleur* d'une instance de la classe *Porte* peut être tirée de celle de son objet composite. Dans le cas présent, la valeur de l'attribut *couleur* de l'instance *Ma-Voiture* n'est pas définie mais est égale à celles de ses composants. Seul le composant *Mon-Capot* a une valeur : celle-ci est propagée à *Ma-Voiture* puis jusqu'à l'instance *Ma-Porte*.

En ce qui concerne les attributs composites multi-valués, les mécanismes d'inférence sont les mêmes pour tous mais peuvent être précisés a posteriori si un éclatement d'attribut a lieu.

L'implémentation de la relation est réalisée grâce aux contraintes. A toute valeur d'une facette de co-héritage est associée une expression équivalente exprimée en terme de contraintes. Ces contraintes sont des contraintes de classe;

elles apparaissent au niveau de la description de l'objet composite. Par exemple, la classe *Voiture* décrite plus haut a une expression équivalente montrée dans la figure 4. L'intérêt des facettes de co-héritage est qu'elles évitent d'utiliser le langage d'expression de contraintes qui, par sa complexité relative, est moins compréhensible.

Voiture

Couleur	**dans**	symbole
Portes	**dans**	Porte
	card	4
Capot	**dans**	Capot

contraintes = {
Affecte(Couleur, Capot.Couleur),
Egal(Couleur, Portes.Couleur)
}

FIG. 4 - : Traduction du co-héritage par les contraintes

En plus de ce mécanisme de partage de valeur, les objets composites bénéficient d'une classification particulière différant légèrement de celle d'objets complexes "classiques".

4.3 La classification d'objets composites

La classification d'un objet composite n'a pas un but différent de celle de n'importe quelle autre instance, à savoir trouver la classe la plus spécialisée auquel l'objet appartient. Néanmoins, on distingue dans le cas des objets composites deux classifications: la première est minimale en ce sens qu'elle ne fait que ce qui est nécessaire pour positionner l'instance à classer dans la hiérarchie de classes (en particulier, la classification des composants est dirigée par la classification initiale). La seconde cherche à classer également les composants aussi bas que possible dans leur hiérarchie propre; ceci peut être très onéreux, c'est pourquoi ce mécanisme doit être explicitement demandé [Marino91].

La section suivante s'attache à décrire le modèle de tâche qui utilise dans sa définition même les objets composites.

5 Modèle de tâches

5.1 Motivations

La notion de tâche est importante dans l'analyse du raisonnement puisqu'elle permet d'exprimer la décomposition d'un problème en sous-problèmes et, par là-même, de décrire les différentes étapes et connaissances nécessaires à sa résolution. C'est pourquoi la tâche constitue dans les travaux sur la modélisation du raisonnement [Steels90, Chandrasekaran86] la brique de base de la conception d'un système à base de connaissances. Cette notion est aussi fortement présente dans les travaux sur l'explication [Jimenez92, David et al.90, Hasling et al.84] puisqu'elle permet à partir de l'analyse d'une trace des tâches exécutées de fournir des explications sur la stratégie utilisée lors de la résolution de problème. Enfin, les travaux sur la reconnaissance de plans [Kautz et al.86, Lin et al.91], ou sur la reconnaissance d'intentions [Nault et al.91], exploitent la notion de tâche (ou d'action) pour confronter aux descriptions de plans existants une trace constituée d'observations dans le but d'en donner une interprétation globale.

L'une des motivations principales du modèle présenté ici est d'offrir un cadre adéquat pour ces trois types d'exploitation de la notion de tâche. Pour ce faire, la tâche est vue comme une entité à part entière que l'on peut décrire (expression de stratégie), construire (exécution), identifier (reconnaissance) et qui peut être exploitée par un raisonnement (explication).

C'est dans cette optique que la notion de tâche est intégrée dans le modèle TROPES en termes d'objets. Ainsi l'existence d'un concept "Tâche" permet d'exprimer et d'organiser à travers une hiérarchie de classes l'éventail des tâches souhaitées. L'exécution d'une tâche est alors ramenée à la construction d'une instance de ce concept. Ainsi, le choix du traitement (décomposition en sous-tâches ou exécution d'une procédure externe) le plus adapté se traduit par le choix de la classe la plus spécifique qui joue alors le rôle de modèle de construction pour l'instance. De même, la décomposition d'une tâche en sous-tâches s'exprime par la relation de composition entre un objet composite et ses composants. Enfin, le flot de données liant les entrées et sorties des différentes tâches se transpose en termes de TROPES par des contraintes d'égalité entre les attributs des instances de tâches.

Dans la suite, la description de la notion de tâche est brièvement donnée tandis que l'expression du flot de données entre tâches en termes de contraintes inter-attributs est détaillée. Pour une présentation plus complète du modèle de tâches et du mécanisme d'exécution on pourra se reporter à [Gensel et al.92].

5.2 La représentation des tâches

5.2.1 Le concept de "Tâche"

Une tâche est décrite en TROPES par une classe du concept prédéfini "Tâche" qui fournit pour la tâche :

- son but. L'attribut 'but' de type chaîne décrit l'objectif que la tâche doit atteindre.
- ses entrées. Elles sont décrites via l'attribut 'entrées', de type ensemble de valeurs, par son éclatement en sous-attributs. Chaque sous-attribut décrit alors une entrée.
- son traitement. L'attribut 'traitement' décrit le lien de relation de composition entre la tâche et ses sous-tâches. Cet attribut est une liste d'objets du concept "Tâche". L'éclatement de cet attribut permet de décrire la séquence de sous-tâches. Quand la tâche n'est pas décomposable mais fait appel directement à une procédure externe, 'traitement' est une liste vide (sa cardinalité est alors fixée à 0).
- ses sorties. Elles sont décrites par l'attribut 'sorties' de la même façon que les entrées.

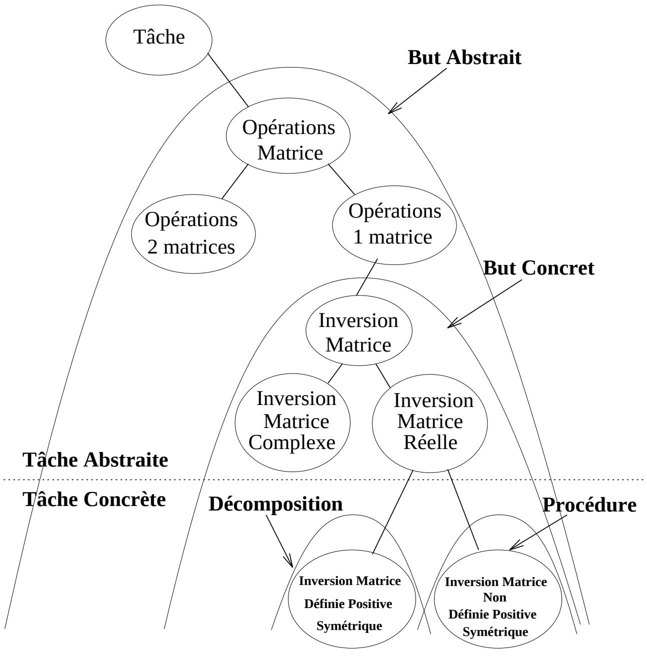

FIG. 5 - : Le concept "Tâche".

L'ensemble des tâches est décrit et organisé par la relation de spécialisation dans la hiérarchie associée au concept "Tâche". L'affinement progressif des descriptions de tâches fait apparaître quatre types de tâche suivant le niveau de précision des divers composants : une tâche abstraite avec but abstrait correspond à la description la plus générale, elle ne précise même pas de but; la tâche abstraite avec but concret indique explicitement un but mais ne fournit pas de traitement particulier; la tâche concrète non-terminale décrit une décomposition en sous-tâches particulière par éclatement des attributs 'traitement' et 'sorties'; enfin, une tâche concrète terminale correspond à un traitement vide et, par conséquent, à un appel de procédure.

La hiérarchie a pour racine la classe 'Tâche' qui regroupe toutes les tâches possibles et représente la tâche abstraite avec le but abstrait le plus général, ensuite les classes sont spécialisées suivant le but, puis les entrées, et enfin le traitement et les sorties. L'allure générale de la hiérarchie est donnée par la Figure 5.

5.2.2 Passage de paramètres entre tâches

La description du flot de données consiste à préciser lors de la décomposition d'une tâche en sous-tâches les diverses liaisons entre paramètres d'entrée et de sortie de ces tâches (cf Figure 6). La transposition de cette description en termes d'objets revient à lier entre eux les (sous-)attributs des attributs 'entrées' et 'sorties' dans la description de la classe décrivant la tâche décomposée. Chacune de ces liaisons est exprimée par une contrainte d'égalité entre les deux (sous-)attributs impliqués. Cette contrainte vient s'inscrire dans l'ensemble des contraintes de classe et assure automatiquement, lors de l'exécution, le passage de la valeur d'un paramètre à l'autre.

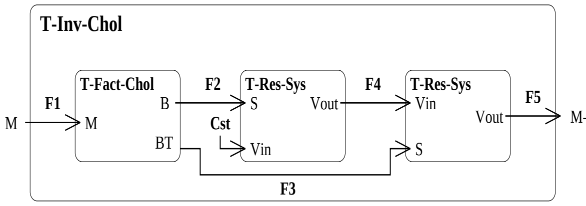

FIG. 6 - : Description du flot de données dans la décomposition de la tâche "T-Inv-Chol" en la séquence de trois sous-tâches "T-Fact-Chol", "T-Res-Sys", puis "T-Res-Sys". 'F1', 'F2', 'F3', 'F4' et 'F5' sont les liaisons entre les entrées et sorties de ces tâches.

Afin de rester proche du formalisme des tâches et de permettre au concepteur d'exprimer le flot de données en termes de liaison entre attributs, une nouvelle facette appelée 'flot' est introduite. Cette facette, qui est suivie d'une chaîne de caractères donnant le nom d'une liaison entre plusieurs attributs, est traduite par le système en contraintes d'égalité. Tous les (sous-)attributs qui ont le même nom de liaison sont contraints à être identiques (cf Figure 7). Cette facilité d'écriture est similaire à celle du partage de propriété pour les objets composites (cf section 4).

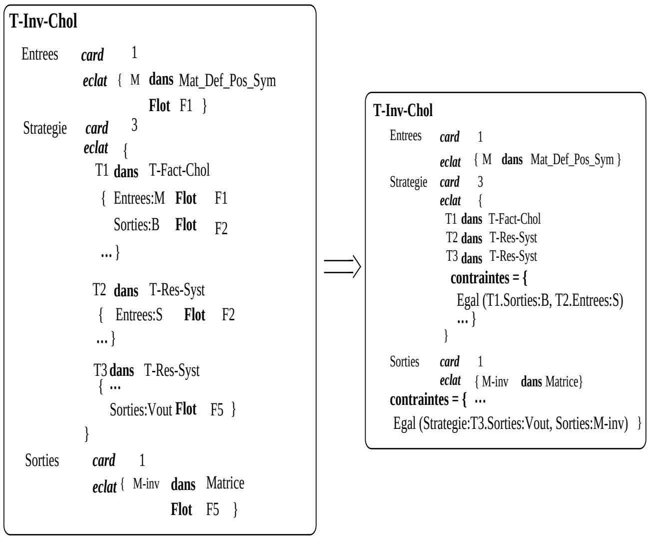

FIG. 7 - : Description du passage de paramètre par la facette 'flot' et son équivalent en termes de contrainte. Par exemple, le flot F1 est réécrit en la contrainte Egal(Entrées:M,Traitement:T1.Entrées:M) où ':' signifie l'accès à un sous-attribut d'un attribut éclaté.

5.3 L'exécution de la tâche

L'exécution d'une tâche correspond à la construction complète d'une instance du concept "Tâche". C'est-à-dire que les quatre attributs 'but', 'entrées', 'traitement' et 'sorties' doivent être connus. Dans le cas contraire, l'instance restant incomplète, l'exécution échoue.

Le processus d'exécution se décompose en trois étapes majeures: la demande d'exécution correspond à l'instanciation d'une classe du concept "Tâche", le but et les entrées étant fournis; si la classe ne caractérise pas une tâche concrète – les attributs 'traitement' et 'sorties' ne sont pas spécifiés – un processus de classification effectue une descente dans la hiérarchie de tâches en fonction des entrées et du but; une fois la classification achevée, si une classe correspondant à une tâche concrète a été trouvée, soit l'instanciation en séquence de chaque sous-tâche décrite dans l'attribut traitement est lancée et les contraintes définissant le flot de données sont activées (cas d'une tâche non-terminale), soit l'exécution de la procédure externe attachée à la classe est lancée (cas d'une tâche terminale).

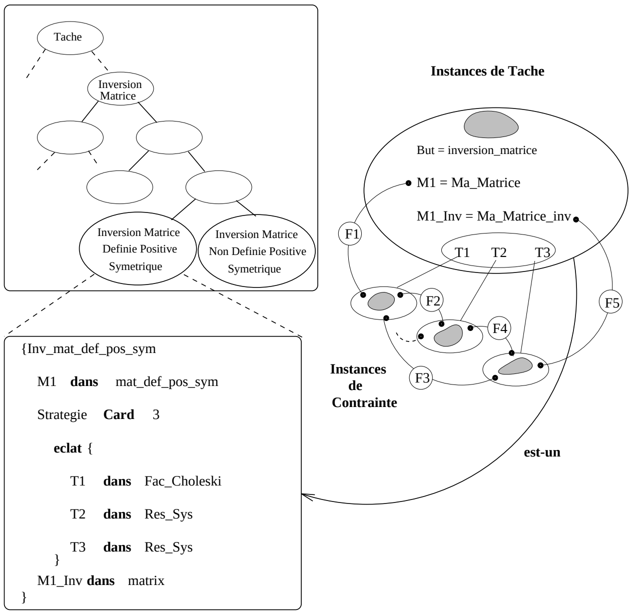

FIG. 8 - : L'exécution d'une tâche. Le rattachement de l'instance à la classe "Inversion de matrice symétrique définie positive" provoque la création de trois instances des tâches apparaissant dans l'attribut 'Traitement', ainsi que l'activation des contraintes entre les attributs de ces instances.

L'exemple (cf Figure 8) montre l'exécution d'une tâche dont le but est d'inverser une matrice. Initialement, la classe "Inversion de Matrice" est instanciée, puis, suite au processus de classification, l'instance de tâche est complétée suivant le moule de la classe "Inversion de matrice symétrique définie positive".

6 Conclusion

TROPES est un système de représentation de connaissances par objets multi-points de vue offrant de puissants mécanismes de raisonnement afin de gérer et de compléter des bases de connaissances. Le travail présenté ici décrit trois extensions successives et complémentaires de ce modèle.

La première extension réside en l'intégration de la notion de contrainte qui permet d'augmenter les capacités de résolution de problèmes du modèle tout en conservant la même logique de représentation. Lors de la définition d'une classe, les contraintes permettent d'exprimer des dépendances entre divers attributs, tandis que le mécanisme de satisfaction de contraintes se charge, d'une part, de garantir l'intégrité des instances appartenant à une telle classe, et, d'autre part, d'inférer dès que possible les valeurs d'attribut manquantes.

La deuxième extension du modèle consiste à enrichir la notion d'objet composite d'un mécanisme de partage de propriétés entre composé et composants qui exploite la transitivité de la relation "partie-tout". Ce mécanisme, appelé co-héritage, est interprété en termes de contraintes.

Enfin, la troisième extension permet au modèle TROPES d'intégrer la notion de tâche dédiée à la représentation de connaissances stratégiques. La tâche est un objet composite particulier décrit dans le concept "Tâche". L'exécution d'une tâche revient à instancier ce concept, à déterminer par classification le traitement à effectuer, et à assurer le passage de paramètres par satisfaction de contraintes.

Nos recherches s'orientent à présent vers une utilisation plus poussée de la notion de point de vue pour les concepts de contraintes et de tâches. L'exploitation d'un raisonnement hypothétique est également à l'étude, qui devrait rendre possible la gestion de mondes possibles en cas de pluralité dans la réponse d'un mécanisme d'inférence. L'instanciation, la classification et l'exécution de tâches devront s'adapter alors à la notion d'instance hypothétique.

Actuellement est en cours de développement une première application de TROPES dans le domaine de la biologie moléculaire [Schmeltzer et al.93]. L'établissement de grandes bases de connaissances destinées à la recherche sur le génome nécessite d'y inclure les connaissances descriptives, méthodologiques, textuelles et graphiques liées à ce domaine scientifique. L'objectif à plus long terme est de fournir aux biologistes moléculaires un environnement coopératif d'analyse de séquences génomiques; cet environnement doit leur apporter, d'une part, des capacités descriptives avancées, des mécanismes de complétion et de caractérisation des entités modélisées (comme par exemple la classification) et, d'autre part, la possibilité de représenter les démarches d'analyse de séquences (grâce à un modèle de tâches adapté).

Références

[Berlandier92] Berlandier (P.). – *Etude des mécanismes d'interprétation de contraintes et de leur intégration.* – Thèse de PhD, Université de Nice, 1992.

[Borning81] Borning (A.). – The Programming Language Aspects of THINGLAB, a Constraint-Oriented Simulation Laboratory. *ACM Transactions on Programming Languages and Systems*, vol. 3, n° 4, 1981, pp. 353–387.

[Brachman et al.85] Brachman (R. J.) et Schmolze (J. G.). – An Overview of the KL-ONE Knowledge Representation System. *Cognitive Science*, vol. 9, n° 2, 1985, pp. 171–216.

[Capponi93] Capponi (C.). – Classification des classes par les types. *Représentations Par Objets (Journée Classification).* – La Grande Motte, France, juin 1993.

[Chandrasekaran86] Chandrasekaran (B.). – Generic Tasks in Knowledge-Based Reasoning: High-Level Building Blocks for expert-System Design. *IEEE Expert*, 1986, pp. 23–30.

[David et al.90] David (J.M.) et Krivine (J.P.). – Structuration du raisonnement et explications. *Revue d'Intelligence Artificielle*, vol. 4, 1990, pp. 77–88.

[FB90] Freeman-Benson (B.). – KALEIDOSCOPE: Mixing Objects, Constraints and Imperative Programming. *1990 Conference on Object-Oriented Programming Systems, Languages and Applications, and European Conference on Object-Oriented Programming.* Ottawa, Canada, pp. 77–88. – octobre 1990.

[Fikes et al.85] Fikes (R.) et Kehler (T.). – The Role of Frame-Based Representation in Reasoning. *Communications of the ACM*, vol. 28, n° 9, septembre 1985, pp. 904–920.

[Gensel et al.92] Gensel (J.) et Girard (P.). – Expression d'un modèle de tâches à l'aide d'une représentation par objets. *Représentation Par Objets.* – La Grande-Motte, France, juin 1992.

[Gensel93] Gensel (J.). – Expression et satisfaction de contraintes dans TROPES, un modèle de représentation de connaissances par objets. *Représentation Par Objets.* – La Grande-Motte, France, juin 1993.

[Hasling et al.84] Hasling (D. W.), Clancey (W. J.) et Rennes (G.). – Strategic Explanations for a Diagnostic Consultation System. *International Journal of Man-Machine Studies*, vol. 20, 1984, pp. 3–19.

[Ilog s.a.92a] ILOG S.A., Gentilly, France. – LE-LISP *Version 16.01, User's Reference Manual*, 1992.

[Ilog s.a.92b] ILOG S.A., Gentilly, France. – PECOS *Version 1.1, User's Reference Manual*, 1992.

[Jimenez92] Jimenez (C.). – L'explicitation des connaissances de contrôle et de stratégie en vue de l'explication. *2èmes Journées Explication du PRC-GdR IA du CNRS*, pp. 131–146. – Sophia-Antipolis, juin 1992.

[Kautz et al.86] Kautz (H. A.) et Allen (J. F.). – Generalized Plan Recognition. *Fifth National Conference on Artificial Intelligence*. – Philadelphia, PA, août 1986.

[Kim et al.89] Kim (W.), Bertino (E.) et Garza (J. F.). – Composite Objects Revisited. *Proceedings of the ACM/SIGMOD International Conference on the Management of Data*. Portland, pp. 337–347. – 1989.

[Kumar92] Kumar (V.). – Algorithms for Constraint-Satisfaction Problems: A Survey. *AI Magazine*, vol. 13, n° 1, 1992, pp. 32–44.

[Lin et al.91] Lin (D.) et Gœbel (R.). – A Message Passing Algorithm for Plan Recognition. *Twelfth Internationl Joint Conference on Artificial Intelligence IJCAI'91*. Sydney, Australia, pp. 280–285. – 1991.

[Marino et al.90] Mariño (O.), Rechenmann (F.) et Uvietta (P.). – Multiple Perspectives and Classification Mechanism in Object-Oriented Representation. *Ninth European Conference on Artificial Intelligence ECAI'90*, éd. par Aiello (L. C.). Stockholm, Sweden, pp. 425–430. – août 1990.

[Marino91] Mariño (O.). – Classification d'objets composites dans un système de représentation de connaissances multi-points de vue. *Huitième Congrès Reconnaissance des Formes et Intelligence Artificielle RFIA'91*, pp. 233–242. – Lyon, France, novembre 1991.

[McGregor et al.92] McGregor (R.B.) et Brill (D.). – Recognition Algorithms for the LOOM Classifier. *AAAI*, pp. 774–779. – San Jose, CA, juillet 1992.

[Montanari74] Montanari (U.). – Networks of Constraints: Fundamental Properties and Applications to Picture Processing. *Information Science*, vol. 7, 1974, pp. 95–132.

[Nault et al.91] Nault (J.P.) et Schmeltzer (O.). – *La reconnaissance d'intentions par appariement d'objets*. – INPG, Grenoble, Rapport année spéciale IA, ENSIMAG, juin 1991.

[PS90] Patel-Schneider (P.F.). – Practical, Object-Based Knowledge Representation Systems. *Information Systems*, vol. 15, n° 1, 1990, pp. 9–19.

[Rechenmann et al.90] Rechenmann (F.), Fontanille (P.) et Uvietta (P.). – SHIRKA*: système de gestion de bases de connaissances centrées-objets, manuel d'utilisation*. – INRIA-IMAG, Grenoble, France, 1990.

[Rechenmann92] Rechenmann (F.). – Method Inheritance and Selection in Class-Based Knowledge Models. *ERCIM Workshop on Theoretical and Experimental Aspects of Knowledge Representation*, pp. 205–208. – Pise, Italie, 1992.

[Schmeltzer et al.93] Schmeltzer (O.), Médigue (C.), Uvietta (P.), Rechenmann (F.), Dorkeld (F.), Perrière (G.) et Gautier (C.). – Building Large Knowledge Bases in Molecular Biology. *First International Conference on Intelligent Systems for Molecular Biology*. Washington, DC. – juillet 1993.

[Steels90] Steels (L.). – Components of Expertise. *AI Magazine*, Summer 1990.

[Winston et al.87] Winston (M. E.), Chaffin (R.) et Herrmann (D.). – A Taxonomy of Part-Whole Relations. *Cognitive Science*, vol. 11, 1987, pp. 417–444.