Integrating Constraints, Composite Objects and Tasks in a Knowledge Representation System

Jérôme Gensel   Pierre Girard   Olivier Schmeltzer

Inria - Rhône-Alpes – Lifia
46, avenue Félix Viallet F-38031 Grenoble Cedex France
Phone: (33) 76.57.45.90. E-mail: Jerome.Gensel@imag.fr

Abstract

Tropes, an object-based knowledge representation system, has been added three important intertwined features: constraints, composite objects and tasks. Constraints provide a declarative means to define and maintain relations 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.

1 Introduction

Knowledge representation systems have proved to be an efficient means to represent both the domain entities and the mechanisms that are going to exploit them.

Tropes is a highly integrated object-based knowledge representation system which handles the semantics of two important relations, specialization and composition. Tropes offers strong problem solving facilities that possess the peculiarity to use the own mechanisms (instantiation, classification, ...) on which the system is based.

After having described the model entities, we will detail its specific features: the constraints, the part-whole relation and the task model.

2 Presentation of the Tropes model

A knowledge base in Tropes [4] is made of independent concepts, each one representing a set of instances. A concept can be seen along many perspectives. In each perspective, classes are organized in a tree. An instance is attached to one and only one class in each perspective (Fig. 1). The class intension

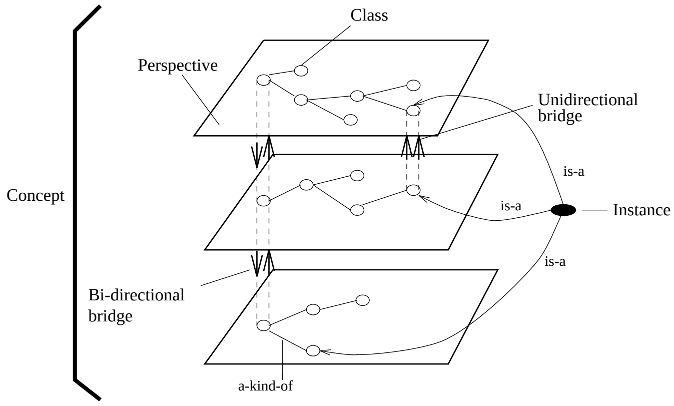

Figure 1: The main entities of the Tropes system

is given by the record of the types of its attributes and ensures that the extension (i.e. the set of instances) of a subclass is contained in the extension of its superclass. These attributes express either a property, a 'part-of' relation link or a relational link. They have a definition for the concept which states their constant features and a definition for each class where they are redefined, thanks to facets, precising a sub-type, the possible and impossible values, the cardinality or the splitting of the attribute into sub-attributes.

A bridge can be thrown over two classes of two different perspectives in a concept, in order to signify a set equality (bi-directional bridge) or a set inclusion (uni-directional bridge) between the extension of the classes.

There are two fundamental mechanims in the Tropes system: instantiation and classification. Instantiation creates an instance, possibly incomplete, by providing its concept, its class in each perspective and values for the attributes satisfying the constraints defined in the concept and in the classes. The classification mechanism allows an instance to migrate from its current class to a more specific one. It takes place in each perspective and checks if the instance satisfies the constraints of the subclasses; this possibly needs to infer some missing attribute values and to use bridges. A class is sure if the instance satisfies all the constraints on all the attributes that exist in this class; it is possible if no constraint in the class is violated, i.e. when the instance is incomplete; it is impossible if at least one constraint is not satisfied by an attribute value.

3 Constraints in Tropes

A constraint is a declarative statement of a mathematical relation which has to be maintained upon a set of variables, each one possessing a domain of possible values. The motivations for integrating constraints in TROPES are twofold. On the one hand, methods are not very convenient for expressing and maintaining mathematical relations between attributes. On the other hand, modeling a relation (for instance a parental relation) by solely representing the relational link restricts the semantics of the relation. If we represent a relation by a TROPES class, we can extend and guarantee its semantics by associating it with a set of constraints (mathematical relations) lying on the attributes of the linked objects.

Four kinds of constraints are available [2]:

A constraint on an attribute indicates a property which must be satisfied by any provided value. It is expressed through a new facet 'cond' which gives the predicate to be invoked.

A constraint on a class expresses a relation between some attributes of the class. Each instance of the class must satisfy this constraint.

A constraint on instances can be selectively set upon some attributes in these instances.

A constraint on a relation involves attributes of the instances linked by the relation.

These three last sort of constraints are modeled by a TROPES class describing the type of each constraint parameter and the function whose invocation will effectively set up and maintain the constraint on the constrained attributes. This invocation is directed by the instantiation of the constraint class. Thus, the setting-up, update or deletion of a constraint is controled by the TROPES classical primitives of object manipulation.

The constraint maintenance is delegated to PECOS [5], a constraint programming library, through an interface which translates any TROPES constraint into an equivalent PECOS constraint, and then transmits to TROPES constrained attributes any change that occurs on the corresponding PECOS constrained variables. Constraint maintenance in PECOS is based on a local propagation which consists in first eliminating impossible values from the domains and then propagating values to infer the missing ones. Once domains have been reduced, a non deterministic programming mode, coupled with a integrated backtrack mechanism, can be used to get the solutions of the given constraint satisfaction problem.

PECOS offers basic functions for defining and manipulating variables of any type on which constraints will be posted (declaration, assignment, domain reduction, ...). A predefined set of constraints (equality, comparison, arithmetic, cardinality, ...) is provided but can be extended by new constraint definitions which mention the conditions (the events that modifies the domain of one of the constrained variables) and the corresponding propagation rules to be invoked.

The TROPES/PECOS interface manages a correspondence table which links any constrained attribute to one or a set of (in case of a multi-valued attribute) PECOS constrained variable(s), controlling the creation, update and deletion of these links. This table stores the reduced domain of a constrained attribute. This interface offers an extensible constraint programming library of functions using access functions to the correspondence table and PECOS functions. Error messages raised by PECOS are signaled to the user (wrong assignment, over constrained attribute).

This integration allows now in TROPES the description of relations in a declarative way. Constraints establish horizontal links between attributes of one or many instances. Thanks to the constraint maintenance performed by PECOS, the consistency of any provided value for a constrained attribute is guaranteed. Moreover, constraints act as an inference mechanism whenever constraint propagation can deduce missing values.

Two other features of TROPES, composite objects and tasks, take also advantage of this descriptive and inference tool as described in the following sections.

4 Composite Objects

The study of the composition relation has shown that it contains a plethora of meanings and could eventually be described as a taxonomy emphasising functional, homeomerous and separable characteristics [7]: the composition relation is *functional* when the parts support a functional role with respect to the whole; it is *homeomerous* when the parts are similar to each other and to the whole and it is *separable* when the parts can be separated from the whole to which they are connected.

The component/integral object meaning has been chosen because it deals with physical objects and it is met often when representing entities.

Four kinds of behavior have been defined according to dependency and sharability [3]. A consequence of the choice of the semantics is that a component cannot be shared by different objects because this situation only occurs with logical objects. With regard to dependency, in TROPES, the destruction of a composite object implies the destruction of its components; however, free components are allowed to exist.

This chosen behavior implies that it is not possible to modify directly a component that belongs to a composite object (i.e. without transiting through it).

Composite objects are very naturally introduced in the TROPES system by specifying different types of attributes: "classical" attributes are used for characterizing entities while composite attributes refer to components. These two types of attributes behave the same way regarding inheritance. As a consequence, a same object can be decomposed in many ways, as many as there are perspectives.

It has been added to the part-whole relation the opportunity to express that the attributes of a composite object and of components can share the same value, or, more generally that the value of an attribute in a composite object is a function (like sum, average, ...) of attribute values of components.

Two new facets have been added to "classical" attributes: 'co-inh?' takes the value 'up' when the attribute value has to be inherited from the composite object, and 'down' when it has to be inherited from the components; 'co-inh-op' specifies the operator to be used to infer the value (Fig. 2). The set of the following operators is predefined but a user is given the opportunity to define his own : equality, addition, minimum, maximum and average.

The direct implementation of the co-inheritance mechanism is achieved through constraints. This notably enables the user to extend the set of sharing operators using the constraint definition capabilities of the TROPES/PECOS interface. For each pair (co-inheritance facet, value), there is an equivalent form in terms of constraints.

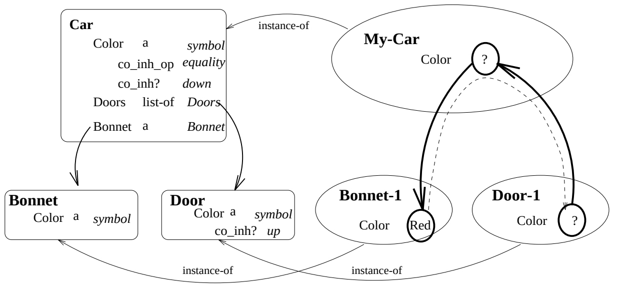

Figure 2: Attribute value propagation. When inferring the attribute value of the *color* of *Door-1*, the system looks for the *color* of *My-Car*; this value is still unknown but can be inferred from the components of *My-Car*; *Bonnet-1* is the only component having its attribute *color* valued; this value is then propagated from the bonnet to the car and finally to the door.

The classification of a composite object posseses peculiar features [4] though its aim is not different from that of any other instance, that is to find the more specialized class the object belongs to. Indeed, two classification mechanisms appear: both intend to put the object as low in the hierarchy of its concept as possible, but, while the minimal classification only checks whether the components satisfy the constraints defined in the composite object class, the maximal classification classifies also the components independently in their hierarchies. Since this second mechanism can be very onerous, the user has to assert it explicitly.

5 The Task Model

A task is an entity in charge of describing the strategy (i.e. the different steps and the knowledge inherent to a reasoning process) for solving a given problem [1, 6]. The task model proposed here emphasizes the production of a trace of the task execution, in the form of objects. The execution trace corresponds to a network where each node, as an object, can be manipulated by the underlying TROPES primitives. A task is a composite object whose components are its sub-tasks. A task class describes the goal of the task, the type of each of its inputs, the strategy (i.e. a list of sub-tasks to be invoked) and the type of its output. The execution of a task comes down to the instantiation of the corresponding composite class.

Task classes are organized in a hierarchy where specialization is achieved on both the inputs and the strategy. A task is *abstract* if its strategy has not been refined. It is *concrete* but *non terminal* if the strategy indicates the sub-tasks to be called. At last, it is a *terminal concrete* task when it is no longer decomposable. A terminal concrete task refers to an external procedure to be invoked.

The parameter passing between a task and its sub-tasks or two successive sub-tasks is expressed thanks to constraints on a task class. When an I/O parameter of a task is shared with one, or more, I/O parameters of another task, an equality constraint is posted between those parameters through a facet 'flow'. This ensures automatically the sharing of the parameter values. The execution of a task is invoked by instantiating the corresponding class. The created instance is incomplete since only the task inputs are provided. The strategy to be adopted and the outputs will be determined by a classification process which moves down the instance along the sub-hierarchy of the initial task class, according to the input. At the end of the classification process, if the strategy is sufficiently described in the most specific task class obtained, then the instantiation of each sub-task described by the strategy is performed; if it is void (the class describes a terminal concrete task), then the execution of the external procedure attached to the task class is performed. Every complete instance of the concept "Task" must be considered a successful execution of the task. If the instance remains incomplete then the task has failed (i.e. there is no strategy for the given inputs, or a procedure cannot be executed or the outputs are invalidated).

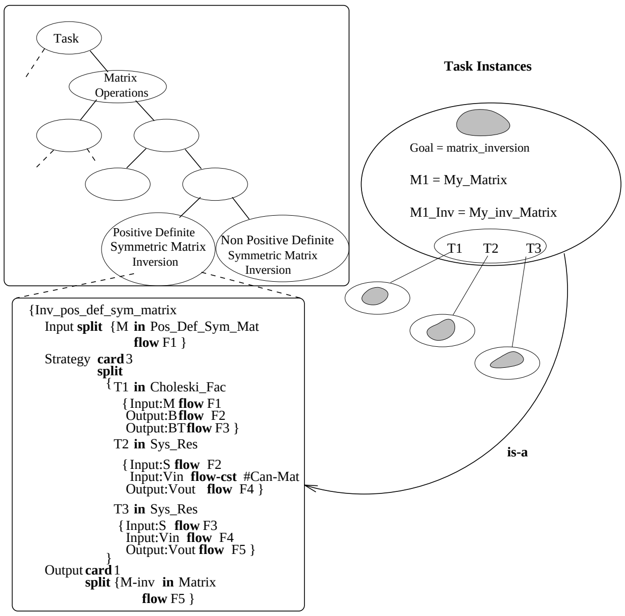

Figure 3: The parameter passing is performed by the facet 'flow' equivalent to an equality constraint. The execution of a task causes the creation of three instances of sub-tasks.

6 Conclusion

Tropes is a multi-perspective object-based knowledge representation system in which the three features presented here – constraints, composite objects and tasks – are fully integrated since they are described by Tropes objects and fully interconnected since constraints are the cement of both the shared properties between composite objects and the parameter passing between tasks.

Our work is now directed towards a farther use of the notion of perspective with regard to the constraints, the relations and the tasks. Another currently work is the study of hypothetical reasoning in order to allow the management of several possible worlds during the use of the inference mechanisms. It leads to adapt the instantiation and classification mechanisms, and the task execution to the hypothetical instance notion.

References

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

[2] J. Gensel. Integrating Constraints in an Object-Based Knowledge Representation System. In *International Conference on Computer Systems and Applied Mathematics*, Saint Petersburg, Russia, July 1993.

[3] W. Kim, E. Bertino, and J. F. Garza. Composite Objects Revisited. In *International Conference on the Management of Data*, volume 2, pages 337–347. Portland, 1989.

[4] O. Mariño, F. Rechenmann, and P. Uvietta. Multiple Perspectives and Classification Mechanisms in Object-Oriented Representation. In *Ninth European Conference on Artificial Intelligence*, pages 425–430. Stockholm, Sweden, August 1990.

[5] J. F. Puget. Programmation par contraintes orientée-objet. In *Conférence scientifique Avignon'92*, pages 129–138, Avignon, France, June 1992.

[6] L. Steels. Components of Expertise. *AI Magazine*, Summer 1990.

[7] M. E. Winston, R. Chaffin, and D. Herrmann. A Taxonomy of Part-Whole Relations. *Cognitive Science*, 11:417–444, 1987.