1

Integrating Constraints in an Object-Based Knowledge Representation System

Jérôme Gensel[1]

ABSTRACT
Object-based knowledge representation systems benefit from the declarative power of the object formalism for modeling, organizing and storing large amounts of knowledge. They also offer powerful reasoning mechanisms (classification, inheritance, methods, ...) which, applied on the current facts of a knowledge base, will deduce implicit or hypothetical facts and then increase the amount of knowledge..
Fully integrating constraints into an object-based representation model should not merely consist in constraining objects managed by the model (allowing the definition of horizontal links between them), but it should also lead to represent constraints with the help of objects (allowing to set, modify or delete them using the basic primitives of object manipulation provided by the model).
Tropes is an object-based knowledge representation system in which constraints are integrated as objects. We describe the main features of the system before detailing the different kinds of constraints available in Tropes. Constraint maintenance in Tropes is delegated to a constraint programming library through an interface we also present here. Constraint propagation serves as an inference mechanism for inferring missing values. Finally, we give some prospects of research offered by this constraint integration.

1.1 Introduction

Based on the principles (expressiveness, class specialization, inheritance, ...) of the Object paradigm, object-based knowledge representation systems are built to describe, organize and store large amounts of knowledge. Moreover, by activating various reasoning mechanisms (like methods, instantiation, classification, ...), they are able to produce new implicit or

[1]**INRIA Rhône-Alpes – Université Joseph Fourier
LIFIA/IMAG, 46 avenue Félix Viallet, 38031 Grenoble Cedex 1, France, E-mail: Jerome.Gensel@imag.fr**

hypothetical facts from the knowledge they contain [14].

As defined in [12, 9], a constraint C($x_1, \ldots, x_n$) describes a relation that has to be maintained on a set V = $\{x_1, \ldots, x_n\}$ of variables (named the constrained variables), each x_i having an associate domain D_{x_i} of values. C can also be seen as a sub-set of the Cartesian product $D_{x_1} \otimes \ldots \otimes D_{x_n}$ which gives all the n-tuples $(v_1, \ldots, v_n)$, where $v_i \in D_{x_i}$, of compatible values for V.

In constraint programming languages, given a constraint satisfaction problem, defined by a set of constraints and the set of their constrained variables, a constraint solver is put in charge of applying satisfaction algorithms to generate, one, some or all the solutions, if they exist. This satisfaction process can be made easier by constraint propagation (which consists in eliminating from each domain, the values that can not appear in any solution) and/or by heuristics (which choose the next constraint to satisfy, the next variable to instantiate, the next value in a domain, ...) [8].

Merging constraints and an object-based knowledge representation system must lead, not only to constrain objects, but also to represent constraints by objects. Constraints can then be manipulated like any other object of a knowledge base.

We describe here the integration of constraints, namely their expression and their maintenance, in an object-based knowledge representation system called TROPES.

While constraint expression is the matter of TROPES entities for knowledge representation, constraint maintenance is delegated to PECOS [7], a constraint programming library of the LE-LISP language [5], thanks to an interface between the two systems. Its role consists in translating any constraint involving TROPES attributes in an equivalent constraint posted on PECOS constrained variables, and, conversely, in transmitting to every constrained attributes the changes performed by PECOS on the corresponding constrained variables.

We present TROPES main principles before exposing the four kinds of constraints available. Then, the tasks performed by the interface with PECOS are stated. Finally, the contributions and the research perspectives stemming from this integration are listed.

1.2 Presentation of the TROPES System

TROPES [11] is a multiple-perspective knowledge representation system which is based on a class instance approach. In TROPES, a knowledge base is made of independent concepts. Each concept models a generic family of the real world and groups a set of instances. Different perspectives can be associated with a concept in order to capture the information particular to each of these views. In each perspective of a concept, classes are organized in a hierarchy and linked by the specialization link ‘a-kind-of’. Each class, except the root of a hierarchy, has got only one superclass and thus inheritance is simple in TROPES. A class provides a description – called the class intention –, which can be seen as a record of the types of its attributes. The set of instances belonging to a class is called the class extension. An instance is attached to one and only one class in each perspective by the instantiation link ’is-a’, but belongs to each class situated on a path from this class to the root (see Fig.1.1). The attribute values of an instance do satisfy the intentions of its belonging classes.

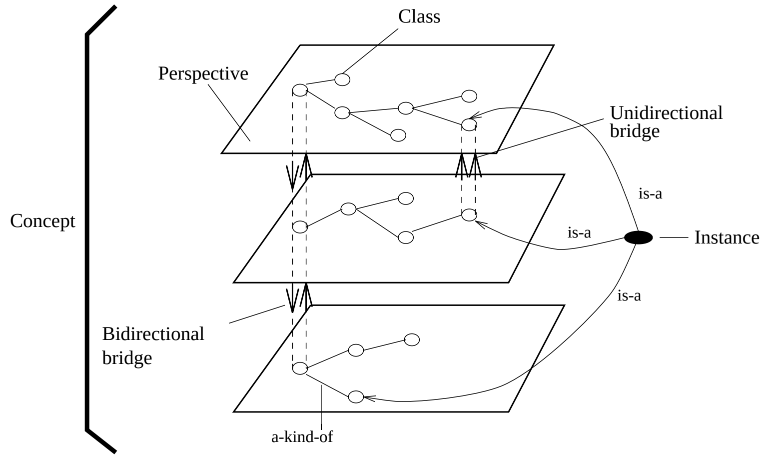

FIGURE 1.1. The main entities of the TROPES system

Each concept determines its proper set of attributes. Acting like a mask, each perspective makes use of only a subset of these attributes (those which are also visible under other perspectives and those which are visible only under this perspective). An attribute expresses either a property, or a ‘part-of’ relation link, or a relational link. It has a definition for the concept which states its general and constant features (type, method, . . .) and a definition for each class where it is redefined. In such a case, the type of the attribute is refined with the help of optional facets precising for the class:

1. Its new type (facet ’in’)
2. The domain of its possible values (facet ’domain’)
3. The domain of its impossible values (facet ’except’)
4. Its cardinality, when the attribute has a value which is a list or a set of values (facet ’card’)

Specialization between a class and one of its direct subclasses is obtained by addition, either of new attributes, or of constraints – specified by the facets presented above – on existing attributes. A subclass (resp. superclass) is said to be more specific (resp. general) than its superclass (resp. subclass). When the description of an attribute is not refined in a class, it is inherited from the superclass.

A bridge can be thrown over two classes (source and destination) of two different perspectives of a concept, in order to signify that a set relation holds between their extensions. A set equality corresponds to a bidirectional bridge, while a set inclusion is described by a unidirectional bridge. These bridges are mainly used during the classification process (described in [11]).

Instantiation is a TROPES mechanism which allows the creation of an instance, possibly incomplete, and its attachment to a class (by default, the root) in each perspective, under the consistency condition that every provided value satisfies the attribute type described in the class intention.

In TROPES, composite objects contain attributes for modeling a 'part-of' relation between them and their parts. In such objects, the values of these attributes are instances – called components – of other concepts. The components are dependent on the composite in both an existential (they were created for it) and exclusive (they cannot be the components of any other composite) way. Classifying an incomplete composite object can lead to the creation of the missing instances of components, according to their description [10].

A strategic knowledge to be applied in a problem solving process can also be described thanks to a task model available in TROPES. A task is a composite object whose instantiation and classification will produce a trace of its execution. This trace can then be used as an explanation of the reasoning process performed.

Finally, a reasoning maintenance system stores the justification and the descendants of any attribute and instance in order to ensure consistency when the knowledge base is modified.

The kernel of the TROPES system encompassing those features has been implemented in the LE-LISP language.

1.3 Constraints Expression in TROPES

1.3.1 MOTIVATIONS

Our motivations for integrating constraints stem from an acknowledgement of a need for expressing relations or properties about objects in TROPES.

Indeed, methods do not reveal themselves convenient when it is a matter of stating a relation between several attributes, mainly because methods are unidirectional while constraints are multi-directional. Only using methods to describe how to satisfy a relation compels the user to associate with each constrained attribute a method which is likely to compute its value from the other attribute values. In this case, the relation is not explicitly expressed but is implicitly represented by a diffuse set of methods. On the contrary, a constraint is a declarative statement which gathers a set of methods dedicated to the maintenance of the relation and which guarantees that the adequate method will be invoked when a change occurs in a constrained attribute. Moreover, using a constraint propagation, impossible values can be eliminated from the domains of constrained attributes. This permits to consider at any moment the effective domain of a constrained attribute and not only the one it has in the class intention. Finally, once a domain has been reduced to a singleton by the constraint propagation phase, it is considered that the possible value for the corresponding variable is unique and known.

In an object-based knowledge representation model, it has seemed to us natural to describe constraints also by objects. Then, the the chosen unit for describing a constraint is a TROPES. This class contains information about the parameters and the code function devoted to the constraint maintenance. The instantiation of such a constraint class corresponds to the effective setting-up of the constraint on the attributes given by the created constraint instance. Since constraints are TROPES objects, the user can define and manipulate them (definition, instantiation or setting-up, modification, deletion) without any change of formalism. On this account the integration performed here is a strong one.

The LE-LISP language, in which TROPES is implemented, has a constraint programming module, called PECOS, which has been used for constraint maintenance. It provides an efficient constraint propagation algorithm based on arc-consistency (or reduction of domain by deleting the impossible values) which is performed on the domain of each constrained variable as soon as the constraint is set and then at any time change occurs in the domain of a constrained variable (for example, when the variable is assigned a value). This library also offers a large range of predefined constraint functions, a module to extend this pool of constraints, functions of constrained variable manipulation and a non deterministic mode for generating the solutions. Using the predefined functions of PECOS allows the definition of arithmetic constraints ($=$, $\neq$, $<$, $>$, $\leq$, $\geq$) on expression made of arithmetic operators ($+$, -, $\times$, $\div$, ...), numbers and TROPES attributes. But, more complex user-defined constraints, or conditional constraints can also be maintained. Although using PECOS spares us the definition of a constraint language, we had to resort to a linking interface for the coupling of the two systems.

Finally, in knowledge representation systems, relations are usually simply represented by attributes which model links between objects. Here again, those (inverse) links are spreaded out in each involved object. Moreover, typing and cardinality facets constitute the only expressible constraints about the nature and the number of the linked objects. It is much more difficult to express in a declarative way, constraints which are induced by the relation, i.e. which must be activated only when the link is established.

Once again, classes reveal themselves suitable description units to capture the characteristics of a relation. Therefore, we propose to represent relations (distinct of the 'part-of' relation) in TROPES by classes which give not only a description of the belonging classes of the linked objects, but also the set of constraints induced by the relation. By this way, the semantics of a relation can be extended and automatically ensured by the constraint solver.

1.3.2 CONSTRAINTS AS UNARY PREDICATES

Typing facets can reveal themselves inadequate when it is a matter of stating a property any value provided for the attribute must satisfy (for instance, being an odd value). For this purpose, we have introduced a new facet 'cond' which indicates a unary predicate representing the property (see Fig.1.2).

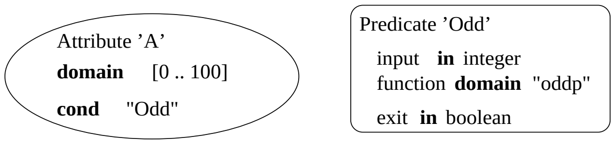

FIGURE 1.2. The value of attribute 'A' has to be an odd value, as it is mentioned by the facet 'cond'. In the predicate class 'Odd', the name of the boolean function to be executed is given. This execution will be activated by the instantiation of this predicate class.

A unary predicate is represented by a class of a concept "Predicates". This class describes the type of the unique argument (attribute 'input'), the code function to be executed in order to determine whether the property is satisfied or not (attribute 'function'), and, the boolean result of the execution (attribute 'exit').

In the presence of a facet 'cond', the checking of a candidate value for an attribute requires the instantiation of the corresponding predicate class. This instantiation activates the execution of the code function. The result is stored in the attribute 'exit'.

1.3.3 CONSTRAINTS AS OBJECTS

In TROPES, a predefined concept "Constraints" is devoted to the definition of constraint classes. A constraint class gives the type of each of its constrained variables, and the function devoted to the constraint maintenance which will effectively post the constraint. This function has only one parameter: the constraint instance (see Fig.1.3) whose attributes are the links between the constrained attributes and the constrained variables used by the constraint programming library PECOS. This function is then written using constraints functions of PECOS (see 1.4.1).

When an instance of constraint is created, if the values of the constrained attributes are all present, TROPES checks for the satisfaction of the relation (local consistency), otherwise it signals the inconsistency. If some attributes have no value, their domain is reduced in accordance with the constraint (arc-consistency). When a domain is reduced to a singleton, this value is propagated (value propagation) for reducing the other domains.

When a constrained attribute is assigned a value by the user for the first time, TROPES checks for its validity (i.e. it must belong to the constrained domain of the attribute, and not only to the domain defined in the class intention). Once it is accepted, this value is propagated: the domains of the other involved attributes are reduced. Whenever one of these domains becomes a singleton, then a value is inferred for the constrained attribute. Therefore, constraints provide a new inference mechanism for TROPES.

When the value of a constrained attribute is modified, its initial reduced domain (the one obtained by the reduction of the class intention domain by the set of constraints lying on the attribute) is restored. If the provided value is in the reduced domain, the status of the previous value is examined. If it was given by the user, its status is kept unchanged and the value is accepted after the modification has been propagated through the attribute descendants by the reasoning maintenance system. Otherwise (i.e. the previous value has been inferred by a constraint), its status must change (the value is now given by the user) as well as the direction of the constraint. In this latter case, one of the adjoining constrained attributes must be invalidated (i.e. its reduced domain is restored), then its new value is inferred. This choice is here made by the user, as we consider that, aware of the modification he or she has confirmed, he or she can may want to control farther the propagation. However, solution maintenance can be automatically performed [1] by an algorithm which generate every possible propagations in reaction to the modification of a constrained variable.

TROPES checks for the validity of the provided value and, if one (or more) of these constraints is no longer satisfied, the inconsistency is raised.

Constraints on Classes

A constraint on a class expresses a relation that holds between some attributes of the class. It can be seen as a property inherent to the class. As a consequence, every instance of this class must satisfy this constraint. For any TROPES class, the set of its constraints is given by a special attribute 'constraints' (see Fig.1.3).

```
Class 'Circle'
attributes = {center_x, center_y,
              radius, area, ...}
constraints = { (Pi-R2 area radius),
              ...}
```

```
Constraint 'Pi-R2'
x in real
y in real
constr-name   domain
              "tr-ct-pi-r2"
```

```
(defun tr-ct-pi-r2 (constr-inst)
    (ct-eq (tr-vcp 'x) (ct-fmul 3.159
              (ct-power  (tr-pcv 'y) 2)
          )
    )
)
```

FIGURE 1.3. In the class 'Circle', the property: area = PI * radius * radius, is imposed by the constraint Pi-R2. The class 'Pi-R2' describes the type of each involved attribute and refers to a function 'tr-pi-r2', written by the user, which is in charge of setting effectively this constraint on each instance of the class. 'ct-eq', 'ct-mult' and 'ct-power' are PECOS predefined functions, while 'tr-pcv' gives the PECOS constrained variable associated with the TROPES constrained attribute.

Concerning TROPES bridges, a constraint lying on the destination class of a unidirectional bridge and involving attributes cwhich are ommon to the source class, must also be applied to the source class of the bridge, since every instance of the source class belongs to the destination class.

Down along the 'a-kind-of' link, the constraints of a class are inherited by its subclasses. If an inconsistency stands between an inherited and a defined constraint, it will be raised by the constraint maintenance system, at any instantiation.

The modification and the deletion of a particular instance of constraint posted on a class are not allowed. Indeed, the constraint being the property of the class, a particular instance of the class can not be released from this constraint separately. Therefore, all the instances of a constraint set on a class are hidden to the user. Only the deletion of the constraint in the set given by the attribute 'constraints' sets the class (i.e. all its instances) free from this deleted constraint.

Finally, since constraints are themselves classes, it is possible to define constraints on constraint classes (meta-constraints) but this characteristic has not yet been explored.

Constraints on Instances

In TROPES, a constraint can also be set dynamically upon attributes by simply instantiating the constraint class which corresponds to the relation which must be maintained. (see Fig.1.4).

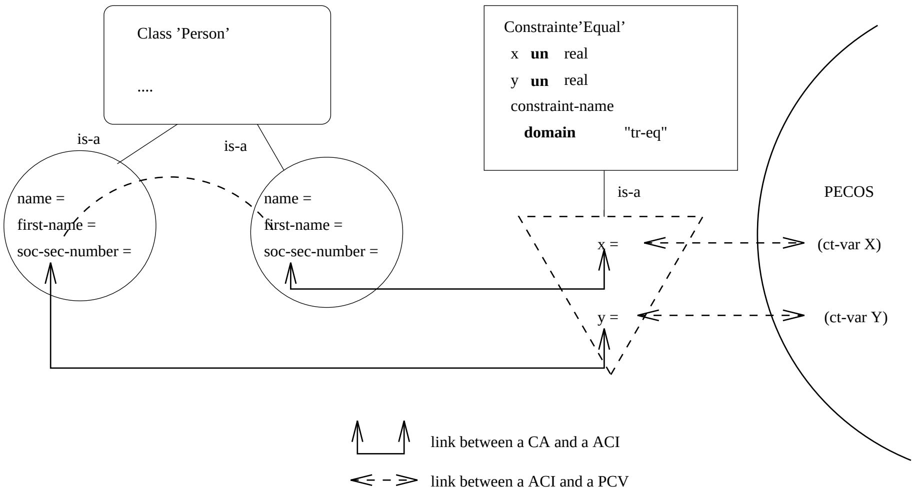

FIGURE 1.4. A unemployed cohabitee can benefit from the Social Security rights of its co-habitant. Then, for the administration, this two persons share the same Social Security number. This is described, in the knowledge base, by an equality constraint on this two particular instances of the class 'Person'. The link between a constrained attribute (CA) and its corresponding attribute in the constraint instance (ACI) and the link between the ACI and the PECOS constrained variable (PCV) are sketched.

As an accessible instance of constraint, a constraint on instances benefits from the other classical operations of manipulation (modification, deletion) provided by TROPES:

1. Modifying the value of an attribute in an instance of constraint comes down to either change the value of the constrained attribute corresponding to the parameter of the constraint, or to modify the value of a constant parameter.
2. Deleting an instance of constraint comes down to release the constrained instances from this link and to propagate the invalidity of any value previously inferred by the deleted constraint.

Constraints of Relations

Relational attributes in TROPES express the possible implication of any instance of a class in a particular relation with instances of another class (we restrict here our study to binary relations). These attributes convey a potentiality of the class: if they have no value, the instance is not (yet) in relation with others.

In TROPES, a relation is represented by a class which describes the two classes linked by the relation (providing in that way the definitions of the two inverse links), and an optional set of constraints associated with the relation (see Fig.1.5. Although the two relation attributes are defined in the relation class, each of them is also visible (copied out) in its corresponding class.

Example: The relation 'Father/Son' is established between the class 'Men' and the class 'Boys' with this particular set of constraints:

The name of a son must be the same as his father's name.

The age of a son must be lower than fifteen years than his father's age.

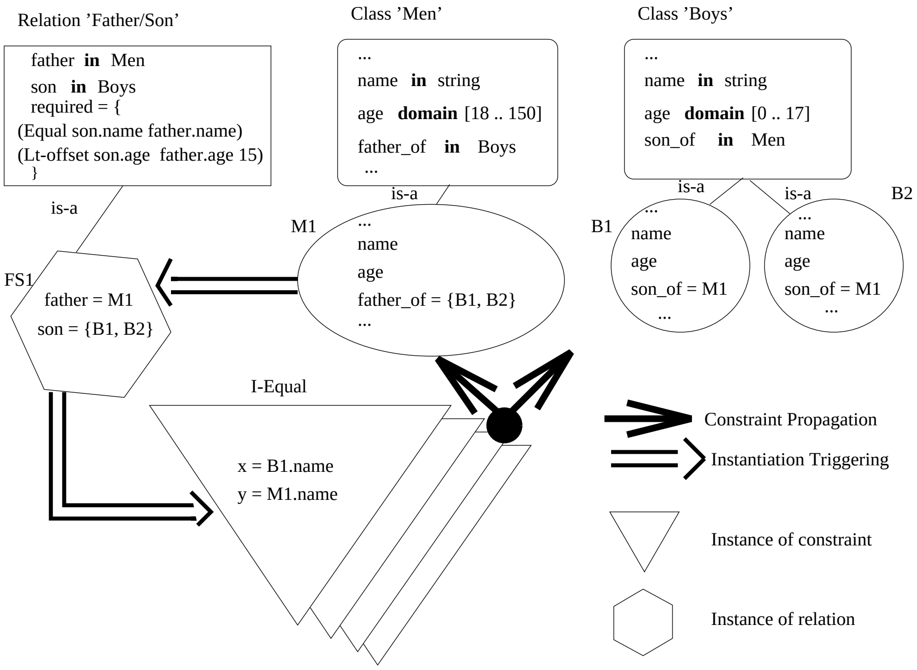

FIGURE 1.5. A relation class 'Father/Son' describes the relation given in the example above. The instance 'FS1' symbolizes the relation that holds between the instances 'M1' and 'B1' and between the instances 'M1' and 'B2'. The instantiated classes of constraints will control the domain of the constrained attributes (namely, 'name' and 'age')

When one of the relational attributes is assigned a value for the first time, the inverse link is updated and an instance of the relation class is created. At its turn, this instantiation triggers off the instantiations of the classes of constraints associated with the relation. Thus, constraints are now fixed on attributes in the newly linked instances, placing them under the control of the relation.

Similarly, the user can decide to instantiate a relation class. Then the values of the new instance are transferred in the corresponding relational attribute in each linked instance. If constraints are associated with the relation, their classes are instantiated.

The modification of an instance of relation aims at extending, modifying or at reducing the set of the linked instances. When an instance is released from its relational link, the value of the relation attribute must be cancelled. Furthermore, the values inferred by any constraint of this relation are no longer valid and a lazy propagation must be activated to report the inconsistency.

The deletion of a relation instance releases the set of the linked instances. This changes must also be propagated.

1.4 Constraint Maintenance in Tropes

1.4.1 The Pecos Constraint Programming Library

The maintenance of the constraints described and set-up in Tropes is handled by Pecos. This Le-Lisp library allows constraint programming with variables of any type over finite domains, with integer variables over enumerated domains, with integer or float variables over an interval, and with set variables whose domain is a set of sets of values.

Pecos offers basic operations for defining and manipulating variables on which constraints will be posted (declaration, assignment, value elimination from the domain, ...). Once constrained variables are defined, predefined constraints (equality, comparison, arithmetic, cardinality, ...) can be posted on them. Moreover, the user can define its own constraints thanks to a provided set of schemas based on a daemon mechanism. A schema mentions a condition (an event that modifies the domain of one of the constrained variables) and a propagation rule to be applied in reaction to this event in order to reduce the domain of the other adjoining constrained variables.

In Pecos, constraint propagation consists in:

1. Eliminating impossible values from the domains as soon as the constraint is posted. For example, let x *in* [0 10] and y *in* [0 5] be two integer variables. The constraint y > x reduces the domains of both x *in* [0 4] and y *in* [1 5].

2. Propagating the modification of a domain to the other domains and, in case of assignment, propagating the value. In the example, if y is assigned the value 1 then x, by reduction of its domain to a singleton, is assigned the value 0.

Pecos proposes also a non deterministic programming mode, coupled with an integrated backtrack mechanism, which can be used notably to perform domain enumerations to get one, many or all of the solutions of the current constraint satisfaction problem.

Although Pecos manages constraints on Le-Lisp objects and classes, this optional object-oriented constraint programming mode is not used by

Tropes, but constraints on classes are treated in the same way (the constraint is uniformly applied to any instance of the class).

1.4.2 The Tropes/Pecos Interface

Although Tropes benefits from the Pecos constraint propagation power, an interface had to be built in order to link every constrained attribute in Tropes to an equivalent (i.e. with the same domain) constrained variable managed by Pecos. The role of this interface consists in:

1. managing a correspondence table which stores every link between a Tropes constrained attribute and a Pecos constrained variable. The interface controls the creation, update and deletion of Pecos constrained variables. Notably, it must be taken into account that the values of Tropes constrained attributes can be modified at any time during the lifetime of a knowledge base, whereas in Pecos, once a domain has been reduced to a singleton, any re-assignment is impossible.

2. offering constraint programming functions for defining and setting-up constraints on Tropes attributes (these functions use access functions to the correspondence table and Pecos predefined functions.

3. adapting the representation of complex domains of Tropes attributes to Pecos. For instance, when a domain is a union of intervals, a composite constrained variable is handled. It corresponds to a list of Pecos constrained variables (each having for domain one element of the union). Thus, a constraint posted on a composite variable is applied to each constrained variable. As soon as one of them has got an empty domain, which means that there is no value in the corresponding interval that satisfies the constraint, the union can be reduced and the constrained variable corresponding to the empty domain is removed from its composite variable.

4. analyzing and reacting to the error messages raised by Pecos. Non deterministic functions are then very useful to control fails and misfortunes and restore a valid state for the Pecos constrained variables (for instance, recovering their previous domain).

1.5 Contributions of this Integration

This integration allows now in Tropes the description of numerical relations in a declarative way and then improves the capabilities of representation of the model. 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 since constraint propagation can deduce missing values.

However, if domains are reduced by arc-consistency, the network of constraints made of constraints on classes, constraint on instances and constraints of relations constitute a constraint network for which the search of solutions is not the main aim in TROPES. For this purpose, the enumerating functions of PECOS could be solicited for generating the solution(s) by enumerating them. This would lead, first to reduce yet more precisely the domains of constrained attributes, and then to the creation of sets of hypothetical instances (a set corresponding to a solution, i.e a set of instances whose constrained attribute values appear in a solution). At present, this integration only deals with consistency maintenance.

Finally, as a descriptive tool, constraints are used by two other main features of the model: composite objects and tasks. Thus, shared properties between a composite object and its components are expressed with constraints and, in the task model, equality constraints define the parameter passing between successive tasks [3].

1.6 Related Works

With regard to the integration of constraints in an object world as objects themselves, the principles presented above are more closely related to the ones advocated in the PROSE [1] or PECOS systems than to the representation of constraints chosen in the THINGLAB [2] or GARNET [13] systems.

In THINGLAB, constraints are relations between the parts of objects. A constraint is defined in a class of objects by a rule (a predicate to test whether the constraint is satisfied or not) and a set of methods (to be invoked to satisfy the constraint). Each time an object is created, the constraints of its belonging class are checked. For each unsatisfied constraint, the system tries to restore the relation by invoking the methods in the specified order. In TROPES, a constraint on classes is referred by the name of the class of constraint to instantiate and the attribute parameter to be constrained. The propagation methods are described by the PECOS constraint function.

TROPES also differs from the GARNET prototype-instance system where constraints are declared inside slots (or attributes) as formulas while relations are expressed by special slots (not classes) and serve two purposes: allowing inheritance of slot values (while it can be refined, if needed, by constraints in TROPES) and automatically creating inverse connections.

PROSE is above all an abstract model of constraints representation and manipulation designed for any object-oriented knowledge representation system though two implementations have been written using [4, 6]. This tool box proposes a general representation for constraints based on three classes: Variable, Constraint and Relation, and offers its own satisfaction and consistency maintenance algorithms. The setting-up of a constraint is also done by instantiating the Constraint class, the resulting instance links instances of the Variable class (objects representing the constrained variables) to an instance of the Relation class (which describes the predicate to satisfy and the inference rules for the values of the constrained variables). In TROPES, the Variable class does not exist because the constraint lies directly on the attributes, while a unique TROPES class of constraint groups the information which is distributed in PROSE in the Constraint and Variable classes. Consistency maintenance in TROPES is ensured by the Reasoning Maintenance System and by the TROPES/PECOS interface. Moreover, PROSE does not allow the declaration of constraints on a class which have to be satisfied by every instance.

As we said, PECOS offers oriented-object programming functions, allowing to constrain an object or a class of a LE-LISP object-oriented programming language. As a TROPES object (instance or class) does not correspond to an object of this language, a straight use of PECOS was not possible. Moreover, in PECOS, the setting-up of a constraint must be planned and performed from the definition of the class by specifying which attributes are susceptible to be constrained. In TROPES, the setting-up of constraints between instances does not impose this condition.

1.7 Future Works

A constraint management system is currently developed above the TROPES kernel.

Our conception work must now be carried out towards both the expression and the management of:

1. usual operators (union, intersection, composition, ...) and properties ((anti) reflexivity, (anti) symmetry, transitivity, ...) on relations, which are now considered as distinct entities in TROPES. These operators address the definition of a relation from existing relations whereas properties can be seen as meta-constraints on relations. This should lead to the integration of an algebraic query language similar to those used in database management systems.

2. control constraints on relations: Let R be a relation between two classes C and C', I and I' instances of respectively C and C'. One might impose that $\forall$ I, $\forall$ I', I R I', or impose a cardinality constraint on the number of I' $\in$ C' such that I R I' holds, etc....

3. constraints between instances of the same class (which is a particular case of a relation):

Let I and I' be two instances of a class C. One might impose that $\forall$ I, $\forall$ I', I $\neq$ I', I.attr Δ I'.attr, where Δ is an operator of comparison

This kind of constraints introduces a notion of order between instances which is a useful feature in a temporal reasoning or in temporal constraints problems.

4. Constraints of evolution which oblige any new attribute value to satisfy a relation with regard to the previous value.
5. hierarchies of relations as it is suggested in [15]. Two semantics can be granted (see Example. 1.3.3)

   The Father/Son relation can be seen as a sub-relation of the Father/Child relation (as a son is a child).

   The Father/Son relation between the 'Men' class and the 'Boys' class can be seen as a sub-relation of the Father/Son relation considered on the unique class 'Human-Male' as the latter is a super class of both the 'Men' class and the 'Boys' class.

   By specialization, the set of constraints associated with a relation should be inherited or reinforced in its sub-relations.

Finally, the interface TROPES/PECOS will have to exploit further the non deterministic programming mode. In particular, a domain enumeration can lead to an attempt of constraint resolution and, if it succeeds, to a hypothetical reasoning by creating the (hypothetical) instances containing the attribute values satisfying the network of constraints, in order to pursue and observe the parallel evolutions of a knowledge base.

1.8 Conclusion

TROPES is an object-based knowledge representation into which constraints are a declarative means for expressing relations both inside and between objects while constraint maintenance offers an inference mechanism based on a value propagation.

The constraint representation is devoted to TROPES classes, allowing the declaration, the setting-up, the modification and the deletion of a TROPES constraint by the classical object manipulation primitives of the model.

The constraint maintenance is delegated to PECOS, a constraint programming library, the trades off (the translation of the TROPES constraints in PECOS ones, the interpretation of the PECOS constraint satisfaction results) are directed by an interface linking the two systems.

This integration extends the range of expressible problems in offering a declarative way to maintain the consistency of new links between objects. It adds a new description level to the class entity in allowing the expression of properties about some attributes which must be satisfied by any instance of the class. Constraints can also be set upon an arbitrary set of instances, at any moment. Finally, composite objects and tasks, two other features of TROPES also benefit from the constraint maintenance.

Our research must now be directed towards both the concept of relation (operators, properties, query language, specialization), and the management of a hypothetical reasoning benefiting from the PECOS non deterministic primitives to explore different configurations (solution spaces) of a knowledge base.

1.9 REFERENCES

[1] P. Berlandier. *Étude de mécanismes d'interprétation de contraintes et de leur intégration dans un système à base de connaissances (in French)*. Phd-thesis, University of Nice Sophia Antipolis, France, 1992.

[2] A. Borning. The programming language aspects of THINGLAB, a constraint-oriented simulation laboratory. *ACM Transactions on Programming Languages and Systems*, 3(4):353–387, 1981.

[3] J. Gensel, P. Girard, and O. Schmeltzer. Integrating constraints, composite objects and tasks in a knowledge representation system. In IEEE Computer Society Press, editor, *Fifth IEEE International Conference on Tools with Artificial Intelligence*, pages 127–130, Cambridge (Boston), MA, USA, November 1993.

[4] ILOG S.A., Gentilly, France. SMECI *Version 1.3, User's Reference Manual*, 1988.

[5] ILOG S.A., Gentilly, France. LE-LISP *Version 15.24, User's Reference Manual*, 1991.

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

[7] ILOG S.A., Gentilly, France. PECOS *Version 1.1, User's Reference Manual*, 1992.

[8] V. Kumar. Algorithms for constraint satisfaction problems: A survey. *AI Magazine*, 13(1):32–44, 1992.

[9] A. K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8(1):99–118, 1977.

[10] O. Mariño. Classification d'objets composites dans un système de représentation de connaissances multi-points de vue (in french). In AFCET, editor, *Huitième Congrès Reconnaissance des Formes et Intelligence Artificielle RFIA'91*, pages 233–242, Lyon, France, November 1991.

[11] O. Mariño, F. Rechenmann, and P. Uvietta. Multiple perspectives and classification mechanisms in object-oriented representation. In L. C. Aiello, editor, *Ninth European Conference on Artificial Intelligence ECAI'90*, pages 425–430. Stockholm, Sweden, August 1990.

[12] U. Montanari. Networks of constraints: Fundamental properties and applications to picture processing. *Information Science*, 7:95–132, 1974.

[13] B. A. Myers, D. A. Giuse, and B. Vander Zanden. Declarative programming in a prototype-instance system: Object-oriented programming without writing methodswhout. *ACM Sigplan Notices*, 27(10):184–200, October 1992.

[14] F. Rechenmann, P. Fontanille P., and P. Uvietta. SHIRKA*: système de gestion de bases de connaissances centrées-objets, User's Reference Manual.* INRIA-IMAG, Grenoble, france, 1990.

[15] J. Rumbaugh. Relations as semantic constructs in an object-oriented language. In *OOPSLA'87*, pages 466–481. ACM, October 1987.