A Pure Graph-Based Solution to the SCG-1 Initiative

J.F. Baget, D. Genest, and M.L. Mugnier

LIRMM (CNRS and Université Montpellier II)
161, rue Ada, 34392 Montpellier - France
{baget, genest, mugnier}@lirmm.fr

Abstract. This paper answers the SCG-1 initiative. The room allocation problem provided has been solved in a *generic* and *automatic* way. The solution is based on a totally *declarative* formal model. Basic constructs are simple graphs and the fundamental operation for doing reasonings is the graph morphism known as projection. The other formal constructs are rules and constraints defined in terms of simple graphs. The modeling framework built upon the formal model allows one to describe a problem with asserted facts, rules representing implicit knowledge about the domain, validity constraints and rules transforming the world. A prototype implementing this framework has been built upon the tool CoGITaNT. It has been used to test our modelization of the room allocation problem.

1 Introduction

Since 1991, our team has been working on CGs along a specific approach [4]. We study CGs as a *graphical* knowledge representation model, where "graphical" is used in the sense of [14], i.e. a model that "uses graph-theoretic notions in an essential and nontrivial way". Indeed, not only CGs are displayed as graphs but also reasonings are based on graph operations. The formal bases of our work are the following. Basic objects are simple CGs, i.e. labeled graphs. The fundamental operation for doing reasonings is the projection, which is a morphism between simple CGs. Reasonings are logically founded, since projection is sound and complete w.r.t. deduction in first order logic. We built extensions of this kernel — namely CG rules [13] and nested CGs [5] — keeping its basic properties. We also developed the software CoGITo [11, 3] (and its extension CoGITaNT [10]), a workbench for building knowledge-based applications, where every piece of knowledge is described by CGs. The theoretical framework and CoGITo have been used in several applications (see [2] and [9] for the applications we are currently involved in).

The Sisyphus-I problem, is a resource allocation task, therefore basically a constraint satisfaction problem. In our previous applications, reasonings were mainly based on deduction (given a knowledge base and a request, find whether the request can be deduced from the knowledge base). Therefore, at least superficially, i.e. in its formulation, the Sisyphus problem is not of same nature as the problems we are used to solve in our framework.

When studying the SCG-1 problem, two immediate questions arose :

– How much could the problem be solved within our framework ? More specifically, how could we represent constraints in our graph-based approach ?
– What additional programming upon CoGITaNT was required ?

The initial requirement for us was to build a solution which operates as much as possible with the existing theoretical framework. The first decision was to keep a completely *declarative* model.

Another decision was to build a *generic* solution, general enough to include at least the family of the "resource allocation task" problems.

A third architectural decision was to design a system able to solve the problem in an *entirely automatic* way. The reason for this choice was not that we did not attach importance to human involvement in the problem solving process. On the contrary, we find it is unrealistic in many constraint satisfaction problems to hope to find a solution satisfying the user without offering him the possibility to add, remove and reformulate constraints during the solving process. But our main goal was to prove that our formal tools were able to solve the problem. Cooperation with the user should come later.

Finally, it seemed important to empirically test our theoretical construction. For that purpose we built a prototype upon CoGITaNT. It is not – yet – a usable tool.

The first section is devoted to the theoretical framework. We then propose a modelization of the Sisyphus problem within this framework. In the last section, we present the prototype and the obtained experimental results.

2 The Formal Model

In order that the paper is self-contained, we will recall all definitions and results needed to understand the modelization, except for the very basic notions (namely basic CGs and their logical translation by the semantics Φ). For more details about the simple graph model see [5]. For rules, see [13, 12]. The notion of a constraint is similar to that of [8] and the notion of a constrained derivation is new. Let us specify that all objects and sets considered here are finite.

2.1 Simple Graphs and Projection

Support The basic ontology is encoded in a structure we traditionally call a *support*. We consider here a simplified version of a support, containing:

– T_C, a partially ordered set of concept types whose greatest element is $\top$;
– T_R, a partially ordered set of relation types. For this problem, we only use binary relations. Each relation type has a signature, which gives the greatest possible concept type for each argument. The partial ordering on relation types may decrease the signature (the more specific a relation type, the more restrictive its signature);

- I, a countably infinite set of individual markers. The following partial order is defined on the set $I \cup \{*\}$, where $*$ is the generic marker: $*$ is the greatest element and elements of I are pairwise non-comparable.

Simple Graphs Basic CGs, without negation or nesting, are the basic constructs. They are used as such to represent asserted facts. They are also the basic structure for more complex constructs such as rules and constraints.

Simple graphs may have co-reference links and difference (or non-co-reference) links. A co-reference link between two nodes says these two nodes represent the same entity. On the contrary, a difference link between two nodes says these nodes represent distinct entities.

Individual concept nodes with the same marker implicitly represent the same entity. Formally we say that two nodes are "co-identical" if either they are related with a co-reference link or they have the same individual marker. Co-identity is an equivalence relation on the set of concept nodes of a graph (see [5]). We say that two nodes are "non-co-identical" if either they are related with a difference link or they have different individual markers.

A simple graph is said to be *valid* if the intersection between the co-identity relation and the non-co-identity relation is empty. In other words, they are no nodes being both co-identical and non-co-identical, nor one node non-co-identical with itself.

CoGITaNT does not implement co-reference and difference links in simple graphs, so we actually simulated these links with two relations types, "equal" and "diff". "equal" is provided with rules saying that it is a reflexive, symmetric and transitive relation [1]. "diff" is provided with one rule saying that it is a symmetric relation. The validity of a graph is defined by means of a constraint, which expresses that two entities cannot be "equal" and "diff" at the same time.

Projection Projection is the basic operation on simple graphs. Its definition is recalled for completeness reasons. A *projection* Π from G to H is a mapping (not necessarily one-to-one) from the nodes of G to the nodes of H which:

1. preserves the graph bipartition (it maps relation nodes to relation nodes and concept nodes to concept nodes);
2. preserves adjacency and order on edges (if a concept node c is the i-th neighbor of a relation node r then $\Pi(c)$ is the i-th neighbor of $\Pi(r)$;
3. may decrease node labels (the order on relation types is that of T_R; the order on concept types is the product of the order on T_C and the order on $I \cup \{*\}$).

Note that the empty graph can be projected into any graph. Since we simulate co-reference and difference links by first-class objects (relations, rules and constraints) previous definition does not take these links into account. Otherwise, some conditions should have been added saying that projection has to preserve co-identity and non-co-identity.

Let us recall that the following results hold when considering the classical logical semantics Φ. Projection is sound with respect to deduction in FOL. It is also complete when the graph H is in normal form, i.e. each node is co-identical only to itself. A graph can be put in normal form by merging co-identical nodes (nodes with the same individual markers or related by co-reference links) provided that they have the same type [5].

2.2 Rules and Derivation

Simple Graphs Rules A rule "If G_1 then G_2" is a couple of simple graphs, G_1 and G_2, respectively called hypothesis and conclusion of the rule, which share some concept nodes. See [13] for precise definitions of a rule, its logical translation, forward and backward chaining operations, and logical soundness and completeness results about these operations. We will use here the notations of [1], which provides a graphical visualization of the rules nicer to see.

In this framework, a *rule* is a simple graph provided with a coloring of its nodes with two colors, say 0 and 1. In drawings, 0-colored nodes are painted in white, and the others in grey. To differentiate rules from the constraints defined below, we mark their graphical representation with the symbol $\boxed{\rightarrow}$. The subgraph induced by the color 0 nodes must be a syntactically correct simple graph. Nodes with color 0 are the hypothesis nodes and make up the *hypothesis* of the rule. Concept nodes of color 0 with at least one neighbor outside the hypothesis part are the *frontier* nodes. Nodes with color 1 are the conclusion nodes (when the rule is applied to a graph in forward chaining, these nodes are added to the graph) and, together with the frontier nodes, they make up the *conclusion* of the rule.

A rule R is *applicable* to a simple graph G if there is a projection, say Π, from the hypothesis of R to G. In this case, the result of the application of R to G following Π is the simple graph G' obtained from G and the conclusion of R by restricting the label of each frontier node c in the conclusion to the label of its image $\Pi(c)$ in G, then joining c to $\Pi(c)$. In this case, we say that G' is *immediately derived* from (G,R). Let $\mathcal{R}$ be a set of rules. G' is said to be *immediately derived* from $(G,\mathcal{R})$ if there exists a rule R in $\mathcal{R}$ such that G' is immediately derived from (G,R).

Derivation and Deduction A graph G' is said to be *derived* from $(G,\mathcal{R})$ if there exists $G = G_0, G_1, \ldots, G_k = G'$ such that each G_{i+1} is immediately derived from $(G_i,\mathcal{R})$. A simple graph H is said *deducible* from $(G,\mathcal{R})$ if there exists a derivation from $(G,\mathcal{R})$ to a graph G' such that H can be projected into G'.

Note that when a rule is applicable to a graph following a projection, it can be indefinitely applied to the resulting graph following the same projection, but all graphs obtained are equivalent to the graph built by the first application. In what follows a rule is applied *only once* to a graph following a given projection. A graph G is said to be *closed* w.r.t. a set of rules $\mathcal{R}$ if all information that can be added by a rule is already present in G (i.e. more formally, for each rule $R \in \mathcal{R}$, each projection from the hypothesis of R into G can be extended to a projection from R as a whole into G). The problem of deciding whether a given graph is deducible from a given knowledge base is a semi-decidable problem. This problem becomes decidable for some specific sets of rules. In particular it is the case when the set of rules is such that every graph can be closed by a finite number of rule applications. In such a case the set of rules is said to be a *finite expansion set.* A *closure* of a simple graph G by a finite expansion set of rules $\mathcal{R}$ is a simple graph derived from $(G, \mathcal{R})$, closed w.r.t. $\mathcal{R}$. All closures are equivalent with respect to projection, and the minimum element of this equivalence class is called *the* closure of G w.r.t. $\mathcal{R}$ and is denoted by $G^*_{\mathcal{R}}$. Fig. 1 presents a graph G

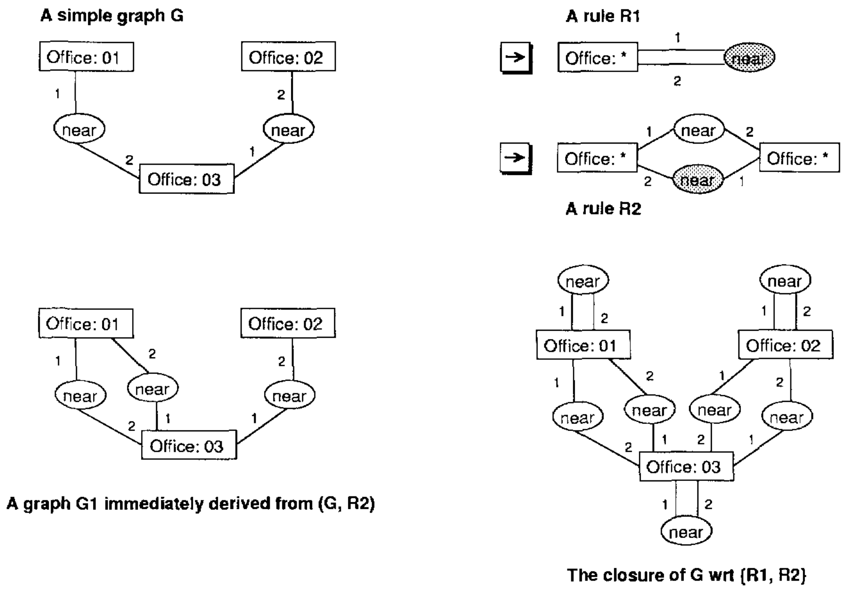

Fig. 1. Applying a rule and closing a graph

expressing that the office #1 is near the office #3 which is itself near the office #2, a rule R_1 expressing that the relation *near* is reflexive, a rule R_2 expressing that the relation *near* is symmetrical, a graph G_1 immediately derived from (G,R_2), and the closure $G^*_{\{R_1 \cup R_2\}}$ of G w.r.t. the rules R_1 and R_2.

For the Sisyphus problem, we use rules in which all concept nodes belong to the hypothesis (in other words the application of a rule only add relation nodes, including the "equal" and "diff" relations). Any set of such rules is obviously a finite expansion set.

2.3 Adding Constraints to the Model

Positive and Negative Constraints We define two kinds of *constraints*: *positive* constraints (very similar to the ones presented in [8]) and *negative* constraints. In the same way as for rules, we will mark positive constraints by the symbol $\boxplus$ and negative constraints by the symbol $\boxminus$. The intuitive semantics that can be attached to these constraints are respectively: "whenever we find the information A in the graph, we must also find the information B", and "whenever we find the information A in the graph, we must not find the information B". Though they have different semantics, their formal definition is identical. A constraint is defined in the same way as a rule, as a simple graph provided with a coloration of its nodes with the two colors 0 and 1. The subgraph generated by nodes whose color is 0 is called the *condition* of the constraint. The condition must be a valid simple graph. In particular, the condition can be an *empty graph*.

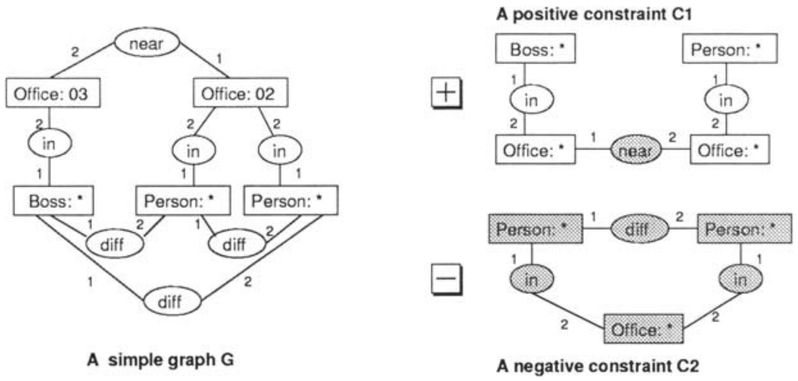

Fig. 2. Example of positive and negative constraints

Let G be a simple graph, and C be a positive constraint. We say that G *satisfies* C if every projection Π of the condition of C into G can be extended to a projection of C as a whole.

Let G be a simple graph, and C be a negative constraint. We say that G *satisfies* C if no projection Π of the condition of C into G can be extended to a projection of C as a whole.

Validity of a Graph Let G be a simple graph, $\mathcal{C}$ be a set of positive and negative constraints. The graph G is said to be *valid* with respect to $\mathcal{C}$ if G satisfies every constraint of $\mathcal{C}$.

The positive constraint C_1 in fig. 2 can be read as "if a boss is in an office and a person is in an office, then these offices must be *near*". The negative constraint C_2 in the same figure can be read as "No office can host two different persons". The graph G in the same figure violates the two constraints :

- For each of the two possible projections of the condition of C_1 into G, we cannot extend the projection to "a good orientation" of the relation *near*.

- The condition of the negative constraint C_2 is the empty graph, which can be projected into G. And this constraint is violated since it exists a projection of C_2 as a whole into G.

Validity of a Graph Given Implicit Knowledge Rules can be used to factorize information, in this case we say that they represent *implicit knowledge*. For the SCG-1 problem, we will restrict these rules to finite expansion sets.

Let G be a simple graph, $\mathcal{R}$ be a finite expansion set of rules representing implicit knowledge, and $\mathcal{C}$ be a set of positive and negative constraints. We say that the knowledge base $(G, \mathcal{R})$ is valid with respect to $\mathcal{C}$ if and only if the closure $G^*_{\mathcal{R}}$ of G is valid with respect to $\mathcal{C}$.

As an example, let G be the graph defined in fig. 2, $\mathcal{R}$ be the set of rules defined in fig. 1, and $\mathcal{C}$ be the set of constraints defined in fig. 2. $(G, \mathcal{R})$ is not valid with respect to $\mathcal{C}$ since $G^*_{\mathcal{R}}$ violates the negative constraint C_2. But also note that $G^*_{\mathcal{R}}$ satisfies now the positive constraint C_1.

Constrained Derivation Consider that *asserted facts* (simple graphs) and *implicit knowledge* rules describe a *world*. Constraints determine whether this world is valid or not. Let us now introduce another set of rules, called *transformation rules*. These rules are used to generate new worlds, which may be valid or not. Given a valid *initial world* $(G, \mathcal{R})$, a set of constraints $\mathcal{C}$ and a set of transformation rules $\mathcal{T}$, answering a request H consists in building a sequence of successive valid worlds issued from the initial one, such that the last one is an answer to H.

Let us give a formal definition of such a derivation.

Let G be a simple graph, $\mathcal{R}$ be a set of rules representing implicit knowledge, $\mathcal{C}$ be a set of positive and negative constraints, and $\mathcal{T}$ be a set of *transformation rules*. We say that a graph G' is a *valid immediate transformation* of G with respect to $(\mathcal{R}, \mathcal{C}, \mathcal{T})$ if and only if:

- $(G, \mathcal{R})$ is valid with respect to $\mathcal{C}$ (i.e. $G^*_{\mathcal{R}}$ is valid in the case of $\mathcal{R}$ being a finite expansion set).
- G' is immediately derived from $(G^*_{\mathcal{R}}, \mathcal{T})$.
- $(G', \mathcal{R})$ is valid with respect to $\mathcal{C}$.

We say that a graph G' is a *valid transformation* of G with respect to $(\mathcal{R}, \mathcal{C}, \mathcal{T})$ if and only if there exists a sequence $G = G_0, G_1, \ldots, G_k = G'$ of graphs such that each G_{i+1} is a valid immediate transformation of G_i with respect to $(\mathcal{R}, \mathcal{C}, \mathcal{T})$. We also say that this sequence is a *constrained derivation* from G to G'.

We say that a simple graph H is *deducible* from $(G, \mathcal{R}, \mathcal{C}, \mathcal{T})$ iff there exists a constrained derivation from G to a graph G' such that H can be projected into G'.

By example, let G be the graph defined in fig. 3, $\mathcal{R}$ be the set of rules defined in fig. 1, $\mathcal{C}$ be the constraints defined in fig. 2, $\mathcal{T}$ be the set of transformation rules containing only the rule R in fig. 3, expressing that we can assign an office

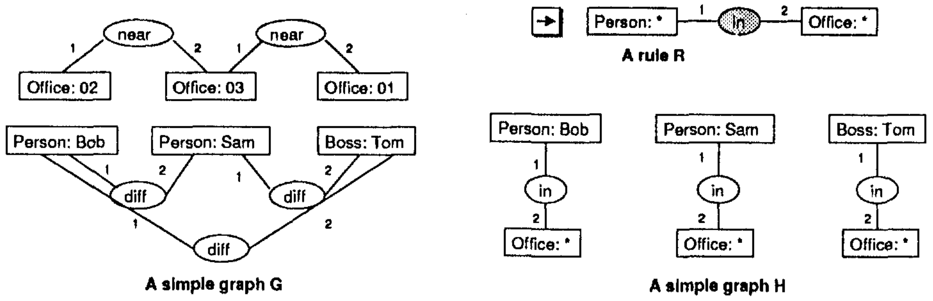

Fig. 3. A graph G, a transformation rule R, and a request H

to each person, and the request H be the graph defined in fig. 3. This request expresses that we want every person to be placed into an office. In order to solve the problem, we first compute the closure of G w.r.t. $\mathcal{R}$, which is a valid graph, then apply the transformation rule once, say by placing *Bob* into the third office. Now, we verify that this graph is valid, and try another assignment, until the request is satisfied. Fig. 4 traces the applications of the transformation rules leading to a correct answer.

```
Tom → 1
        Sam → 1 Violation: two in the same office
        Sam → 2 Violation: far from boss
        Sam → 3
                Bob → 1 Violation: two in the same office
                Bob → 2 Violation: far from boss
                Bob → 3 Violation: two in the same office
Tom → 2
        Sam → 1 Violation: far from boss
        Sam → 2 Violation: two in the same office
        Sam → 3
                Bob → 1 Violation: far from boss
                Bob → 2 Violation: two in the same office
                Bob → 3 Violation: two in the same office
Tom → 3
        Sam → 1
                Bob → 1 Violation: two in the same office
                Bob → 2 Solution found
```

Fig. 4. A trace of the backtrack leading to the solution

3 Representing the Problem

In this section, we present the support, fact graphs, implicit knowledge rules, constraints, transformations rules and the request we used to model and solve the Sisyphus-I problem.

3.1 The Support

The support defined in fig. 5 and 6 represents all types used in our modelization. Relation types "equal" and "diff" represent co-identity and its negation. Each relation type in fig. 6 is provided with its signature.

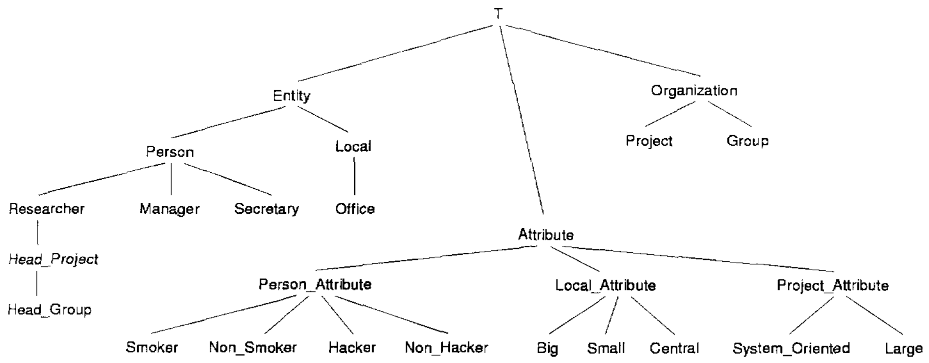

Fig. 5. Hierarchy of concept types

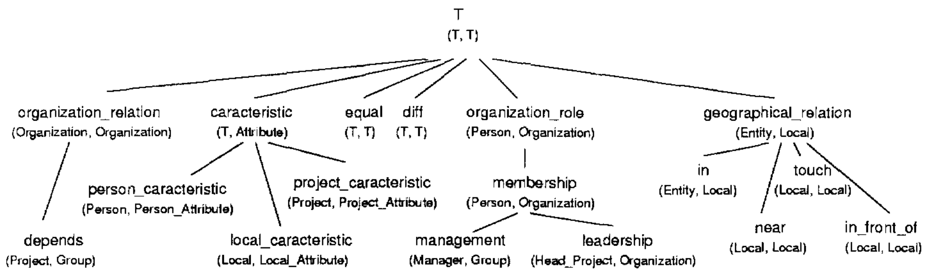
T (T, T)
organization_relation (Organization, Organization)
caracteristic (T, Attribute)
equal (T, T)
diff (T, T)
organization_role (Person, Organization)
geographical_relation (Entity, Local)
person_caracteristic (Person, Person_Attribute)
project_caracteristic (Project, Project_Attribute)
membership (Person, Organization)
in (Entity, Local)
touch (Local, Local)
depends (Project, Group)
local_caracteristic (Local, Local_Attribute)
management (Manager, Group)
leadership (Head_Project, Organization)
near (Local, Local)
in_front_of (Local, Local)

Fig. 6. Hierarchy of relation types

3.2 Fact Graphs

The initial graph is obtained by making the disjoint union of the graphs represented in figs. 7, 8, 9 and 10. These graphs have been separated for better readability. To ensure completeness of our computation, we had the choice between using the rules assigned to the *equal* relation, or to compute the *normal form* of the resulting graph, by merging concept nodes having the same individual marker. The latest solution was chosen for a performance purpose. To present more readable graphs, we did not represent *diff* relations in these graphs. In the graph of fig. 7, all pairs of concept nodes typed *Office* and having different individual markers are linked by a relation node typed *diff*. The same assumption is done for the concept nodes typed *Project* in graph of fig. 8, and for concept nodes whose type is more specific than *Person* in graph of fig. 10.

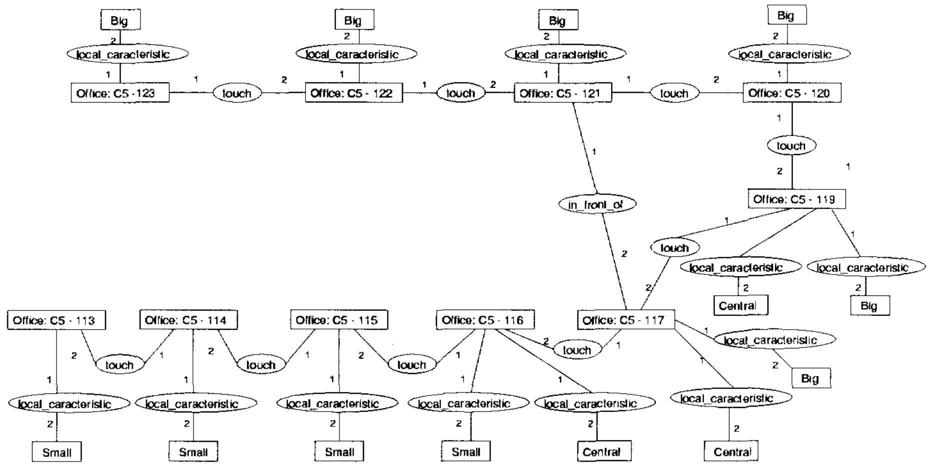

Fig. 7. Geographical information for the first floor of the château of HNE

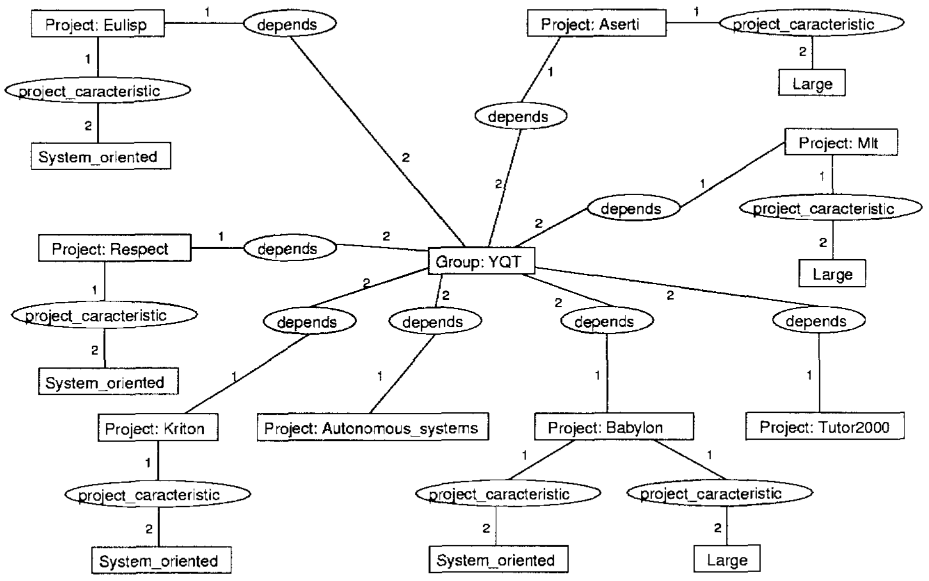

Fig. 8. Organizational structure of the YQT group

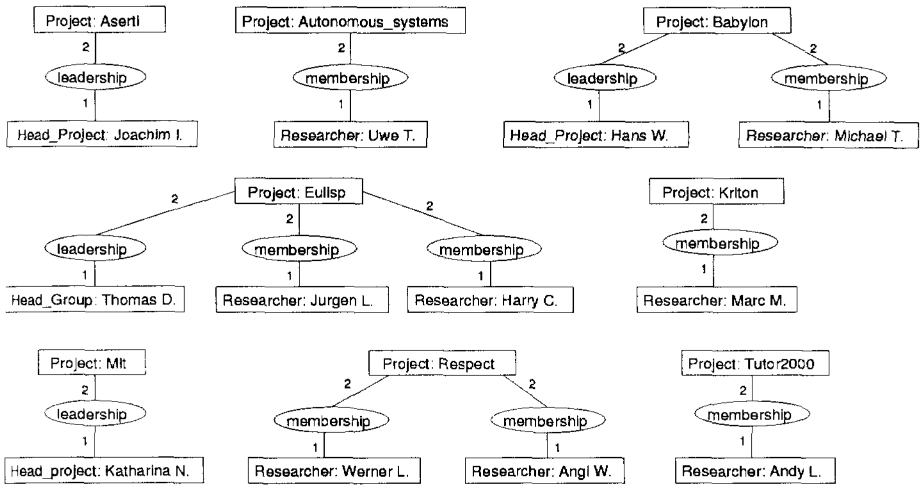

Fig. 9. Team members of the YQT group

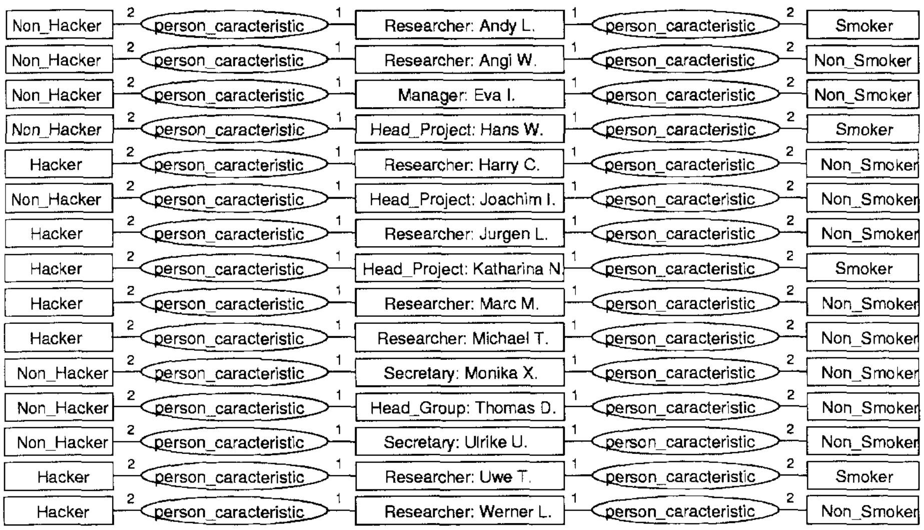
Non_Hacker — person_caracteristic — Researcher: Andy L. — person_caracteristic — Smoker
Non_Hacker — person_caracteristic — Researcher: Angi W. — person_caracteristic — Non_Smoker
Non_Hacker — person_caracteristic — Manager: Eva I. — person_caracteristic — Non_Smoker
Non_Hacker — person_caracteristic — Head_Project: Hans W. — person_caracteristic — Smoker
Hacker — person_caracteristic — Researcher: Harry C. — person_caracteristic — Non_Smoker
Non_Hacker — person_caracteristic — Head_Project: Joachim I. — person_caracteristic — Non_Smoker
Hacker — person_caracteristic — Researcher: Jurgen L. — person_caracteristic — Non_Smoker
Hacker — person_caracteristic — Head_Project: Katharina N — person_caracteristic — Smoker
Hacker — person_caracteristic — Researcher: Marc M. — person_caracteristic — Non_Smoker
Hacker — person_caracteristic — Researcher: Michael T. — person_caracteristic — Non_Smoker
Non_Hacker — person_caracteristic — Secretary: Monika X. — person_caracteristic — Non_Smoker
Non_Hacker — person_caracteristic — Head_Group: Thomas D. — person_caracteristic — Non_Smoker
Non_Hacker — person_caracteristic — Secretary: Ulrike U. — person_caracteristic — Non_Smoker
Hacker — person_caracteristic — Researcher: Uwe T. — person_caracteristic — Smoker
Hacker — person_caracteristic — Researcher: Werner L. — person_caracteristic — Non_Smoker

Fig. 10. Personal data about members of the YQT group

3.3 Implicit Knowledge – the Set of Rules $\mathcal{R}$

The five rules represented in fig. 11 express that:

A. The relation *near* is symmetrical.
B. Two locals that *touch* the same one are considered *near.*
C. The relation *touch* is symmetrical.

D. Two locals that *touch* each other are considered *near.*
E. The same for locals being *in_front_of* each other.

Note that the two last rules could be replaced by expressing that *in_front_of* and *touch* are two relation types more specific than *near.* We did not represent, but use the rules expressing that *in_front_of* and *diff* are symmetrical.

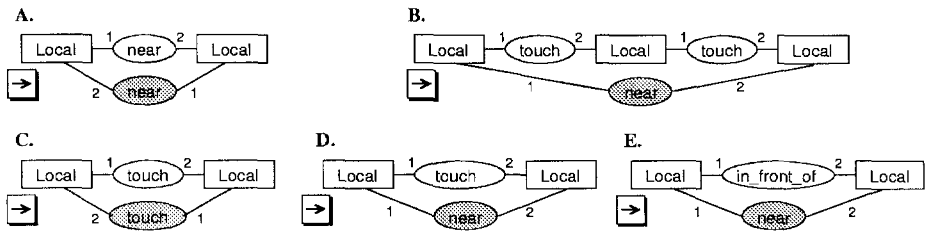

Fig. 11. Rules representing implicit knowledge

3.4 Constraints – the Set *C*

The constraints related to possible assignments can be either positive or negative. We also sorted these constraints in three categories of decreasing priority: absolute, strong and weak. It is not possible to violate an *absolute* constraint. *Strong* constraints are to be satisfied by any solution, but, if the problem is over-constrained, the user may accept to reformulate it by modifying this set of constraints. *Weak* constraints represent preferences.

Every constraint used is described below, but whenever we have very similar constraints, only one of them is represented.

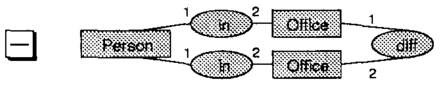

Fig. 12. Ubiquity ?

The graph represented in fig. 12 is a negative constraint. It expresses that a person cannot be into two different places at the same time. This is an *absolute* constraint.

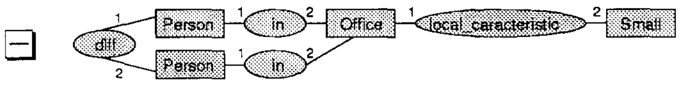

Fig. 13. Number of persons in small offices

The graph represented in fig. 13 is a negative constraint. It expresses that a small office cannot host more than one person. This is a *strong* constraint.

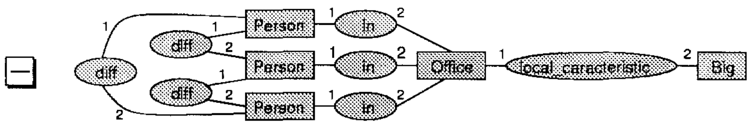

Fig. 14. Number of persons in big offices

The graph represented in fig. 14 is a negative constraint. It expresses that a big office cannot host more than two persons. This is a *strong* constraint.

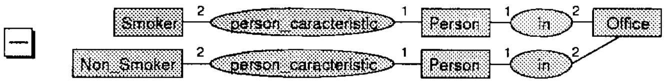

Fig. 15. Smoker – Non-Smoker antagonism

The graph represented in fig. 15 is a negative constraint. It expresses that no office can host a *Smoker* and a *Non_Smoker*. This is a *strong* constraint.

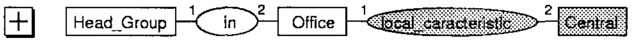

Fig. 16. Head of group accessibility

The graph represented in fig. 16 is a positive constraint. It expresses that the head of group needs a central office (if the head of group is in an an office, then this office has to be a central one). This is a *strong* constraint. In the same way, the manager would be pleased to have a central office, but we consider this as only a *weak* constraint.

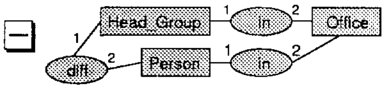

Fig. 17. Privileges of the head of group (1)

The graph represented in fig. 17 is a negative constraint. It expresses that the head of group needs to be alone in his office. This is a *strong* constraint. In the same way, the manager as well as heads of large projects would be pleased to be alone in their office, but we consider this as only *weak* constraints. We also added a positive constraint expressing that the head of a large project should be in a small office. This *weak* constraint is a consequence of the others, and is present for optimization purposes.

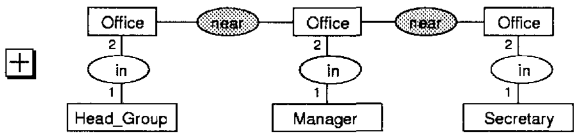

Fig. 18. Manager's neighborhood

The graph represented in fig. 18 is a positive constraint. It expresses that the manager needs to be near the head of group as well as the secretariat. This is a *strong* constraint. In the same way, heads of large projects should be close to the head of group as well as the secretariat, but we only consider this as only a *weak* constraint.

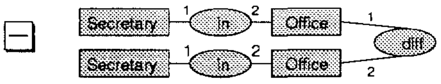

Fig. 19. Secretariat holds secretaries

The graph represented in fig. 19 is a negative constraint. It expresses that no two secretaries can be in different offices. Note that this constraint can be satisfied since there are only two secretaries. Otherwise, should we want "secretaries-only" offices, we could express it by a positive constraint: "if a person is in the same office as a secretary, this person must also be a secretary". This is a *weak* constraint.

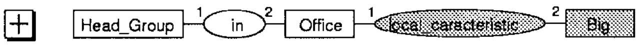

Fig. 20. Privileges of the head of group (2)

The graph represented in fig. 20 is a positive constraint. It expresses that the head of group would like to have a big office. This is a *weak* constraint.

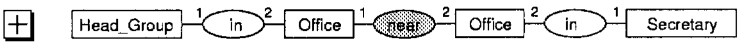

Fig. 21. Secretariat's accessibility

The graph represented in fig. 21 is a positive constraint. It expresses that the secretaries' office should be located close to the office of the head of group. This is a *weak* constraint.

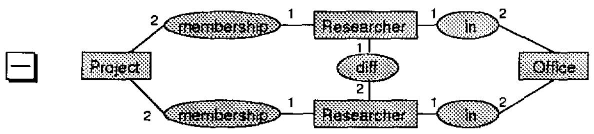

Fig. 22. Synergy

The graph represented in fig. 22 is a negative constraint. It expresses that members of a same project should not share offices. This is a *weak* constraint.

3.5 Transformations – the Set of Rules $\mathcal{T}$

There is only one rule adding information to the base graph, it is the rule represented in fig. 23 which tries to assign a given person into a given office.

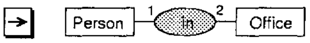

Fig. 23. Placing a person into an office

3.6 The Request

The graph we want to deduce is the solution to the SCG-1 problem: we want every person being placed into an office. This graph is represented in fig. 24.

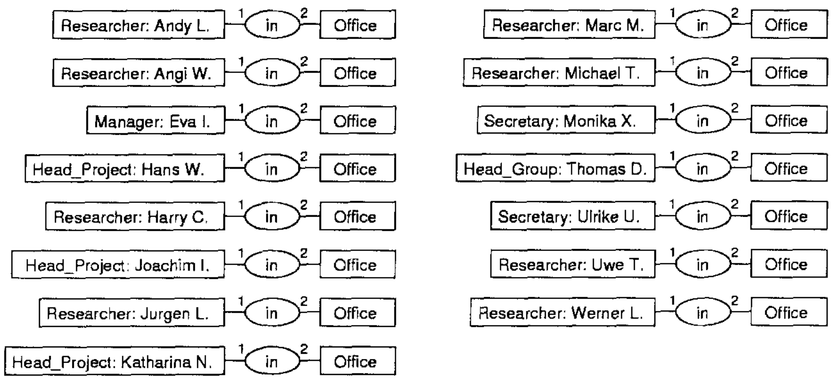

Fig. 24. The request

4 The Evaluation Process

This section first presents some considerations about the computational complexity of the solving process. A deeper study of this complexity is yet to be done. We then present the implementation of the *Constrained Derivation Engine* on top of CoGITaNT and results of the computation.

4.1 Combinatorial Considerations

The existence of a projection is a NP-complete problem [6]. The existence of a deduction from a simple graph G and a set of rules $\mathcal{R}$ is a semi-decidable problem [7] (by analogy with TGDs). In case of a finite expansion set of rules, the problem is obviously decidable. In particular, when the rules only have relation nodes in their conclusion, the problem is NP-complete.

The problem of knowing if a graph is valid with respect to a set of constraints is a co-NP-complete problem (as we need to exhibit two projections in order to prove that a constraint is violated).

If the set of rules $\mathcal{R}$ representing implicit knowledge is a finite expansion set, then the problem of deduction with constrained derivation is semi-decidable. If the set of rules $\mathcal{T}$ representing the possible transformations of the world is also a finite expansion set, then the problem is decidable. This is the case in our modeling of the Sisyphus-I problem.

We consider now the tree representing the possible applications of the transformation rule. For each step, we have to choose between *# of persons* $\times$ *# of offices = 150* possible assignments of a person in an office. There are *# of persons = 15* such steps, and this lead us to the study of the validity of 150^{15} different worlds. Even if we consider that violations of the constraints will prune many branches of this tree, this is not reasonable.

So we have chosen, for this problem only, to force a particular person to be assigned at each step, and this choice leads to *only* 10^{15} possible worlds. We also forced these assignments to follow a specific order, which is more or less the one suggested by the wizard Siggi D. These restrictions, however incomplete in a general case, are well-founded in the case of our modelization of the SCG-1 problem. We are currently studying the possibility of using heuristics to compute automatically the best order with respect to the rules and constraints involved.

4.2 Implementation of the Constrained Derivation engine

We developed constrained derivation on top of the CoGITaNT platform [10]. Our work is yet a prototype library and not a definitive tool. CoGITaNT can be seen as a tool for manipulating graphs, computing projections and applying rules, all graphs involved being read in the BCGCT format [11]. As can be seen in fig. 25, our work is based upon a client–server architecture. A graphical editor has been designed, which communicates with the *Editor Server* built on top of CoGITaNT. This tool allows the user to edit and modify simple graphs, rules and constraints which form the knowledge base located on a distant server.

We have also added on top of CoGITaNT a constrained derivation engine, which is not yet provided with the necessary optimizations. This engine is intended to communicate with a user-friendly *Problem Manager* client, which will enable the user to select rules and constraints that he/she needs for a particular problem solving and visualize a step by step evolution of the base graphs by interacting with the graph editor. This *Problem Manager* is not yet ready, and we had to hard-code the SCG-1 problem on top of the constrained derivation engine.

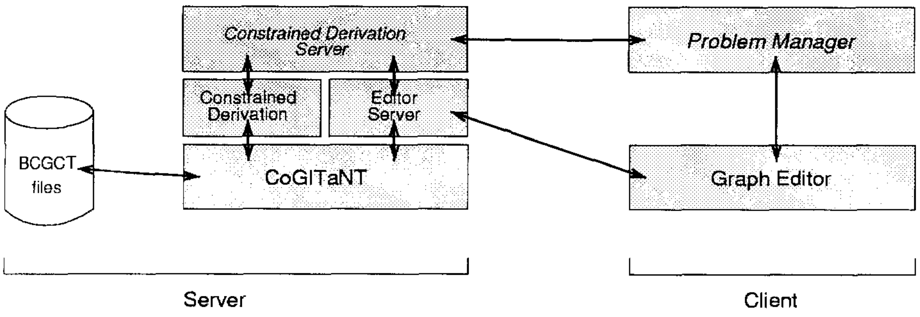

Fig. 25. System architecture

In order to develop a generic constrained derivation engine, we defined several new classes built on top of the the C++ classes of the CoGITaNT library. These new classes are a straightforward implementation of the model presented in section 2, and no particular work has yet been done to optimize the research process.

- The class `rule`, which provides methods such as the application of a rule on a graph given a projection, has been updated to compute the closure of a graph with respect to a single rule. The class `set_of_rules` mainly provides a calculus of the closure of a graph by a (finite expansion) set of rules.
- The class `constraint` represents positive and negative constraints and a method of this class verifies whether a given graph satisfies the constraint. The class `set_of_constraints` determines whether a given graph is valid with respect to this set of constraints.

These classes are used to define the core of the engine. The class `cd_problem` allows the user to add to the problem the BCGCT files containing the fact graph[1], rules representing implicit knowledge, constraints, transformation rules and the request. Each graph is added to the problem in the same way, we just have to indicate the nature of the graph contained in the file such as "negative weak constraint" or "transformation rule". Once these graphs have been loaded, the method `execute` can be called and computes *one* or *every* solution.

Of course, this engine can be easily used on the modeling described in section 3. The `scg1_problem` class, inheriting from `cd_problem` benefits from the optimization presented in section 4.1.

To take into account the different priorities assigned to constraints, we used the following algorithm. We sorted all constraints used along a total order such

[1] A single graph or several graphs can be loaded. In the latter case, the fact graph is automatically computed as the normal form of the disjoint sum of these graphs.

that we have ***weak*** constraints first, then ***strong*** constraints, then ***absolute*** constraints. If we find a solution satisfying every constraint, then we return this solution. Otherwise, we remove the weakest constraint from C and try to find a solution to this modified problem. If we can find a solution by removing only weak constraints, we answer "yes, there is a solution if we remove the following constraints ...". If the only solution is obtained by removing at least one strong constraint, then we answer: "no, there is no solution unless you accept to remove the following constraints ...". If there is still no solution after having removed all weak and strong constraints, then we answer "there is no solution to the problem".

4.3 Solutions Found

The research engine found 2880 solutions to the SCG-1 problem. These solutions satisfy all constraints excepted a weak one which expresses that heads of large projects should be next to the secretariat and the head of group. Obviously, this constraint cannot be satisfied. Given our modeling, the number of solutions can be explained as follows:

- The head of group can only be in the office 117 or 119 *(2 solutions)*
- The manager can only be placed in a small central office, i.e. the office 116. *(1 solution)*
- The positions of the head group and manager determine the position of the secretariat (the office which has not been assigned to the head of group) *(1 solution)*
- There are only three small offices left, and they must be assigned to the three heads of large projects, which should be alone in their office *(6 solutions)*
- The two smokers must be together and have the choice between the 4 big offices left *(4 solutions)*
- There are 15 possible sets of couples for the 6 researchers left, of them 5 are impossible (they are in the same project). There are 6 possible assignments for these couples in the 3 last big offices *(60 solutions)*

Finally, we have the $2 \times 1 \times 1 \times 6 \times 4 \times 60 = 2880$ different solutions exhibited by our research engine. Two of the computed solutions are presented in fig. 26. The first solution we find is the solution A in fig. 26. The beginning of the research tree used to find this solution is presented in fig. 27. We can see there one backtrack, as the first secretary is assigned a small office (no explicit constraint forbids it), and problems arise only when we try to assign the same small office to the second secretary. We explored 85 different worlds before generating the first valid solution (so we had 70 backtracks), and we must remind here that each of these worlds required to solve the problem of its validity, which is a co-NP-complete problem. However, the length of the computation is counted in seconds.

Office	Solution A
113	Hans W. (Head of Large Project)
114	Katharina N. (Head of Large Project)
115	Joachim I. (Head of Large Project)
116	Eva I. (Manager)
117	Thomas D. (Head of group)
119	Monika X. Ulrike U. (Secretaries)
120	Andy L. Uwe T. (Researchers, smokers)
121	Michael T. Mark M. (Researchers, non smokers)
122	Jurgen L. Werner L. (Researchers, non smokers)
123	Angi W. Harry C. (Researchers, non smokers)

Office	Solution B
113	Katharina N. (Head of Large Project)
114	Hans W. (Head of Large Project)
115	Joachim I. (Head of Large Project)
116	Eva I. (Manager)
117	Monika X. Ulrike U. (Secretaries)
119	Thomas D. (Head of group)
120	Angi W. Mark M. (Researchers, non smokers)
121	Harry C. Werner L. (Researchers, non smokers)
122	Jurgen L. Michael T. (Researchers, non smokers)
123	Andy L. Uwe T. (Researchers, smokers)

Fig. 26. Two solutions to the SCG-1 problem

```
Thomas D. → 113 Violation: Not Central
Thomas D. → 114 Violation: Not Central
Thomas D. → 115 Violation: Not Central
Thomas D. → 116 Violation: Small Office
Thomas D.→ 117
        Eva I. → 113 Violation: Not Central
        Eva I. → 114 Violation: Not Central
        Eva I. → 115 Violation: Not Central
        Eva I. → 116
                Monika X. → 113 Violation: Far from Manager
                Monika X. → 114 Violation: Far from Manager
                Monika X. → 115
                        Ulrike U. → 113 Violation: Far from Manager
                        Ulrike U. → 114 Violation: Not with other Secretary
                        Ulrike U. → 115 Violation: Two persons in a small office
                        Ulrike U. → 116 Violation: Two persons in a small office
                        Ulrike U. → 117 Violation: Head of group must be alone
                        Ulrike U. → 119 Violation: Not with other Secretary
                        [ ... ] (120, 121, 122, 123) Violation: Not with other Secretary
                Monika X. → 116 Violation: Two persons in a small office
                Monika X. → 117 Violation: Head of group must be alone
                Monika X. → 119
                        Ulrike U. → 113 Violation: Far from Manager
                        Ulrike U. → 114 Violation: Not with other Secretary
                        Ulrike U. → 115 Violation: Not with other Secretary
                        Ulrike U. → 116 Violation: Two persons in a small office
                        Ulrike U. → 117 Violation: Head of group must be alone
                        Ulrike U. → 119
                                Hans W. → ...
```

Fig. 27. A trace of the backtrack leading to the solution A

4.4 Coping with Changes in Data

In order to cope with the slightly modified data set, we only have to modify the base graph and the request for our problem.

- The graph represented in fig. 9 (team membership) is modified by removing `[Head_Project:Katharina N.]→(leadership)→ [Project:Mlt]`, replacing it by `[Researcher:Christian I.]→(membership)→[Project:Mlt]`
- The graph represented in fig. 10 (personal information) is modified by removing the connected component containing the concept node whose marker is `Katharina N.` and replacing it by: `[Hacker]←(person_caracteristic)← [Researcher:Christian I.]→(person_caracteristic)→[Smoker]`.
- The request presented in fig. 24 is modified in such a way that the concept node `[Head_Project:Katharina N.]` is replaced by the concept node `[Researcher:Christian I.]`.

As we have less constraints about Christian I. than about Katharina N., there are now 8640 different solutions to the problem (once again, according to our modelization). As in the first problem, the last weak constraint cannot be satisfied.

If the objective is to modify as few as possible the existing solution, one can proceed in the following way. Weak constraints can be added, expressing the fact that everybody (or some persons) would like to stay at the same place. This is done by transforming each or some of the previous assignments in a negative constraint, such as the one represented in fig. 28. These constraints express that a person should not be in an office different from the one he/she has already been assigned to.

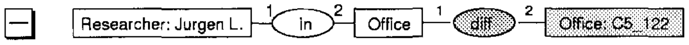

Fig. 28. Minimum displacement constraint (case of Jurgen L.)

By introducing all such constraints in our problem (nobody want to change), it is not surprising that the only one solution found replaces Katharina N. by Christian I.

Conclusion

The fundamental objective of the Corali project [3] is to develop conceptual graphs as a graphical knowledge representation model (in the sense given in the introduction of this paper). The research works are based on a four-stroke experimental methodology: build a theoretical formal model, build software tools implementing the formal model, use the two preceding points to build real-world applications, then evaluate the systems built, and loop through this four-step process.

This paper presents a *generic* way of solving the Sisyphus-I room allocation problem, in the sense that the modeling framework introduced is general enough to enable the representation of any resource allocation problem. This framework is based on a *graphical* formal model. Basic constructs are simple conceptual graphs and the basic operation for doing reasonings is projection. Two more complex constructs are defined in terms of simple graphs: rules and constraints. They are processed by operations based on projection. To summarize the different kinds of knowledge represented by these constructs, we can say that :

- asserted *facts* provided with *implicit knowledge* about the domain define *worlds*;
- *constraints* express conditions for a world to be considered as valid;
- *transformation rules* define possible changes in worlds;
- *the request* represents a question.

Answering a request consists in finding a *constrained derivation* from the initial world to one satisfying the request.

Using the modeling constructs to produce a modelization of the given problem is done in a rather "natural" way: simple graphs assert facts using an initial vocabulary limited to primitive types (concepts and relations) and individuals, rules represent implicit knowledge resulting from the asserted facts, constraints translate obligations and interdictions for possible solutions of the problem in a rather straightforward way, and transformation rules describe the way solutions are constructed.

Let us add that all CG applications we were involved in seem to confirm that the formal constructs are really understandable and usable by an end-user. Graphs in their graphical form are easy to read and operations are not difficult to understand because they are "matching" operations and can be graphically represented.

Dealing with the modified set of data was straightforward: we only had to slightly modify asserted facts and the request. Using constraints we were also able to specify which existing room assignments should be kept in the new solutions.

A *prototype* implementing this framework has been built upon CoGITaNT. It has been used to test our modelization of the Sisyphus-I problem. Constraints have been classified into three clusters of different priority, allowing more flexibility in the answers given by the system. This prototype is not yet a really usable tool. Further developments are needed in order to enable communication with the user in a friendly way, give the user the possibility to intervene at several stages of the solving process, and improve computational complexity of the solving process.

References

1. J.-F. Baget. A simulation of co-identity with rules in simple and nested graphs. In *Proceedings of the 7th ICCS*, 1999.
2. C. Bos, B. Botella, and P. Vanheeghe. Modeling and simulating human behaviors with conceptual graphs. In *Conceptual Structures: Fulfilling Peirce's Dream, ICCS'97 Proc., LNAI 1257*, pages 275–289. Springer Verlag, 1997.
3. B. Carbonneill, M. Chein, O. Cogis, O. Guinaldo, O. Haemmerlé, M.L. Mugnier, and E. Salvat. The COnceptual gRAphs at LIrmm Project. In *Proc. of the first CGTOOLS workshop*, pages 5–8, 1996.
4. M. Chein. The CORALI project: From conceptual graphs to conceptual graphs via labelled graphs. In *Conceptual Structures: Fulfilling Peirce's Dream, ICCS'97 Proc., LNAI 1257*, pages 65–79. Springer Verlag, 1997.
5. M. Chein, M.-L. Mugnier, and G. Simonet. Nested Graphs: a Graph-based Knowledge Representation Model with FOL semantics. In *Proc. KR'98*, pages 524–534. Morgan Kaufmann, 1998.
6. M. Chein and M.L. Mugnier. Conceptual Graphs: Fundamental Notions. *Revue d'Intelligence Artificielle*, 6(4):365–406, 1992.
7. S. Coulondre and E. Salvat. Piece Resolution: Towards Larger Perspectives. In *Conceptual Structures: Theory, Tools and Applications, ICCS'98 Proc., LNAI 1453*, pages 179–193. Springer Verlag, 1998.
8. J. Dibie, O. Haemmerlé, and S. Loiseau. A Semantic Validation of Conceptual Graphs. In *Conceptual Structures: Theory, Tools and Applications, ICCS'98 Proc., LNAI 1453*, pages 80–93. Springer Verlag, 1998.
9. D. Genest and M. Chein. An experiment in Document Retrieval Using Conceptual Graphs. In *Conceptual Structures: Fulfilling Peirce's Dream, ICCS'97 Proc., LNAI 1257*, pages 489–504. Springer Verlag, 1997.
10. D. Genest and E. Salvat. A Platform Allowing Typed Nested Graphs: How CoGITo Became CoGITaNT. In *Conceptual Structures: Theory, Tools and Applications, ICCS'98 Proc., LNAI 1453*, pages 154–161. Springer Verlag, 1998.
11. O. Haemmerlé. *CoGITo : une plate-forme de développement de logiciels sur les graphes conceptuels*. PhD thesis, Université Montpellier II, 1995.
12. E. Salvat. Theorem proving using graph operations in the conceptual graph formalism. In *Proceedings of the 13th European Conference on Artificial Intelligence (ECAI'98), Brighton, UK*, 1998.
13. E. Salvat and M.L. Mugnier. Sound and complete forward and backward chainings of graph rules. In *Conceptual Structures: Knowledge Representation as Interlingua, ICCS'96 Proc., LNAI 1115*, pages 248–262. Springer, 1996.
14. L.K. Schubert. Semantic Networks are in the Eye of the Beholder. In J. F. Sowa, editor, *Principles of Semantic Networks*, pages 95—108. Morgan Kaufmann, 1991.