List of Figures

List of Tables

Goal Modelling and Reasoning in Tropos

Paolo Giorgini[1], John Mylopoulos[1,2], and Roberto Sebastiani[1]

[1]Department of Information and Communication Technology

University of Trento – Italy

{Paolo.Giorgini, Roberto.Sebastiani}@unitn.it

[2]Deparment of Computer Science

University of Toronto – Canada

jm@cs.toronto.edu

Abstract. We overview on-going research on modelling and analyzing goals in requirements engineering. Specifically, we introduce and discuss a qualitative model for goals, which – among other things -- can cope with qualitative relationships and inconsistencies among goals. We then present an axiomatization of the model and propose sound and complete algorithms for forward and backward reasoning. In particular, given a goal model and labels for some of its goal elements, forward reasoning focuses on how to propagate these labels forward, towards root goals. Backward reasoning, on the other hand, focuses on finding a label assignment for leaf nodes of a goal graph that collectively satisfy/deny all root goals. Assuming that the satisfaction/denial of any leaf goal requires some unit cost, we also address the problem of finding a minimum cost label assignment to leaf goals for satisfying/denying all root goals of a goal graph. Both problems are solved by reducing them to the problems of satisfiability (SAT) and minimum-cost satisfiability (minimum-cost SAT) for Boolean formulas. The proposed algorithms have been implemented and are available through a tool named the GR-tool. To illustrate the whole framework, we include a simple case study adopted from the literature where a goal model is built and then analyzed.

1. Introduction

One of the distinguishing elements of *i** is its use of goals to ascribe intentions to actors. The modelling and reasoning framework adopted for goals is derived from the softgoals of the NFR framework [3, 14]. This chapter revisits this framework and proposes a revision that is well-founded, both semantically and algorithmically. The revision constitutes one of the three formal reasoning techniques supported by the Tropos methodology for developing agent-oriented software systems [1,2]. Specifically, goals are used in Tropos to model and analyze functional and non-functional requirements for the system-to-be, also to represent dependencies among stakeholders and components of the system-to-be.

The concept of goal has been used in different areas of Computer Science since the early days of the discipline. In AI, problem solving and planning systems have used the notion of goal to describe desirable states of the world [15]. For example, a planning system might be given the goal "on(A,B) and on(B,C)", which describes states where blocks A, B, C form a stack. The planning system can then analyze the goal (e.g., by decomposing it into two subgoals) and find suitable actions that will satisfy it. For this setting, goal analysis consists of decomposing goals into subgoals through an AND- or OR-decomposition. If goal G is AND-decomposed (respectively, OR-decomposed) into subgoals G_1,G_2…G_n, then all (at least one) of the subgoals must be satisfied for the goal G to be satisfied. Given a goal model consisting of goals and AND/OR relationships among them, and a set of initial labels for some nodes of the graph (S for "satisfied", D for "denied") there is a simple label propagation algorithm that will generate labels for other nodes of the graph. The propagation is carried out from subgoals towards an AND/OR-decomposed. This algorithm can be used to determine if the root goals of a goal model are satisfied, given an assignment of labels for some of the leaf goals.

Unfortunately, this simple framework for modelling and analyzing goals won't work for domains where goals are used to represent the intentions behind design decisions [6, 14]. In such domains, goals can't always be formally defined, and the relationships among them can't be captured by semantically well-defined relations such as AND/OR ones. For example, goals such as “Highly reliable system" admit no formal definition that prescribes their meaning for all stakeholders, though one may want to define necessary conditions for such a goal to be satisfied. Moreover, such a goal may be related to other goals, such as “Thoroughly debugged system", “Thoroughly tested system" in the sense that the latter obviously contribute to the satisfaction of the former, but this contribution is partial and qualitative. In other words, if the latter goals are satisfied, they certainly contribute towards the satisfaction of the former goal, but don't guarantee it. The framework will also not work in situations where there are contradictory contributions to a goal. For instance, we may want to allow for multiple decompositions of a goal G into sets of subgoals, where some decompositions suggest satisfaction of G, while others suggest denial. The use of goals in Goal-Oriented Requirements Engineering is thoroughly surveyed and discussed in [11, 17].

This paper overviews our own work on goal modelling and analysis, already presented in earlier papers [8, 9, 10, 18]. Specifically, we introduce and discuss a formal model for goals, which – among other things -- can cope with qualitative relationships and inconsistencies among goals. We then presented an axiomatization of the model and propose sound and complete algorithms for forward and backward reasoning. In particular, given a goal model and labels for some of its goal elements, forward reasoning focuses on how to propagate these labels forward, towards root goals. Backward reasoning, on the other hand, focuses on finding a label assignment for leaf nodes of a goal graph that satisfies/denies all root goals. Assuming that the satisfaction/denial of any leaf goal requires some unit cost, we also address the problem

of finding a minimum cost label assignment to leaf goals for satisfying/denying all root goals. Both problems are solved by reducing them to problems of satisfiability (SAT) and minimum-cost satisfiability (minimum-cost SAT) for Boolean formulas.

To give an intuitive idea of the approach, consider the simple goal model represented in Figure 1. The figure shows a single root goal “Protect users" that might be associated with a public transit system. This goal is AND/OR decomposed several times. The figure also includes some positive qualitative contributions, e.g., “Protect driver health" contributes positively (“+" label) to the goal “Ensure driver capabilities". Forward reasoning considers as input labels for some of the lower goals of the goal model, for instance “provide rules”, “check capabilities”, “check attitudes” and “check “health”, and infers labels for goals higher up through propagations from the AND/OR subgoals to a parent goal, also propagations in the forward direction for qualitative relationships. Conversely, backward reasoning is given labels for some root goals, such as “protect users” and “protect drivers health” and looks for an assignment of labels to leafs that can generate the desired assignment for top goals.

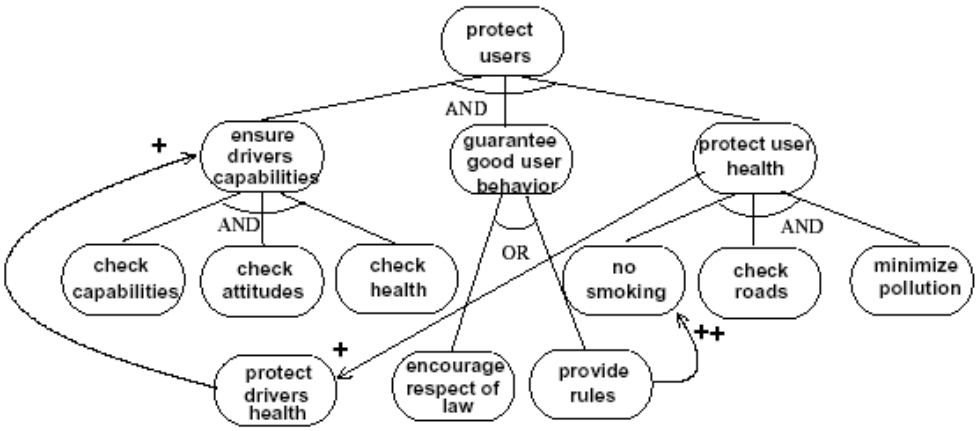

Figure 1. A simple goal graph

At the present forward and backward reasoning support only the requirements analysis phase of the Tropos methodology. In particular, goal reasoning is applied during the early requirements phase to analyze stakeholders goals and possible interactions and

conflicts among the activity of each single actor. During the late requirements phase, forward and backward reasoning is applied to evaluate alternatives requirements during the phase of requirements selection. Moreover, the reasoning support the analyst in finding and solving possible conflicts among requirements. A more detailed description of the use of our approach in the Tropos methodology can be found in [8].

The rest of the paper is structured as follows. Sections 2 defines goal graphs and proposes an axiomatization for goal relationships. Sections 3 and 4 formulate respectively the problems of forward and backward reasoning for goal graphs and propose algorithms for solving them. Section 5 demonstrates the use of goal analysis using a case study from the literature, while section 6 describes a goal analysis tool that has been implemented to support formal reasoning with goal models. Section 7 concludes.

2. Goal Models

In this section we present the formal model for goals adopted in Tropos. The model supports the representation of qualitative relationships and inconsistencies among goals. The following sections introduce the notion of a goal graph and propose an axiomatization for goal relationships.

2.1. Goal Graphs

We consider sets of goal nodes G_i and of relations $(G_1,...G_n) \xrightarrow{r} G$ over them, including the (n+1)-ary relations AND/OR and the binary relations +S, -S, +D, -D, ++S, --S, ++D, --D, +, -, ++, --. We briefly recall the intuitive meaning of these relations:

- $(G_1, ...,G_i, ...G_n) \xrightarrow{and} G$ means that G is satisfied [resp denied] if all $G_1, ..., G_n$ are satisfied [resp. if at least one G_i is denied];
- $(G_1, ...,G_i, ...G_n) \xrightarrow{or} G$ means that G is denied [resp satisfied] if all $G_1, ..., G_n$ are denied [resp. if at least one G_i is satisfied];

- $G_2 \overset{+S}{\rightarrow} G_1$ [resp. $G_2 \overset{++S}{\rightarrow} G_1$] means that if G_2 is satisfied, then there is some [resp. a full] evidence that G_1 is satisfied, but if G_2 is denied, then nothing is said about the denial of G_1;
- $G_2 \overset{-S}{\rightarrow} G_1$ [resp. $G_2 \overset{--S}{\rightarrow} G_1$] means that if G_2 is satisfied, then there is some [resp. a full] evidence that G_1 is denied, but if G_2 is denied, then nothing is said about the satisfaction of G_1.
- $G_2 \overset{-D}{\rightarrow} G_1$ [resp. $G_2 \overset{--D}{\rightarrow} G_1$] means that if G_2 is denied, then there is some [resp. a full] evidence that G_1 is satisfied, but if G_2 is satisfied, then nothing is said about the denial of G_1;
- $G_2 \overset{+D}{\rightarrow} G_1$ [resp. $G_2 \overset{++D}{\rightarrow} G_1$] means that if G_2 is denied, then there is some [resp. a full] evidence that G_1 is denied, but if G_2 is satisfied, then nothing is said about the satisfaction of G

The names +S, -S, +D, -D, ++S, --S, ++D, --D have the following intuitive meaning: "S" [resp. "D"] symbol denotes the fact that the satisfiability [resp. deniability] value of the source goal is propagated; the "+" [resp. "-"] symbol denotes the fact that the propagation is positive [resp. negative], in the sense that satisfiability propagates to satisfiability [resp. deniability] and deniability propagates to deniability [resp. satisfiability].

The meaning of or, +D, -D, ++D, --D is dual w.r.t. and, +S, -S, ++S, --S respectively. (By "dual" we mean that we invert satisfiability with deniability.) The relations +, -, ++,-- are defined such that each $G_2 \overset{r}{\rightarrow} G_1$ is a shorthand for the combination of the two corresponding relationships $G_2 \overset{rS}{\rightarrow} G_1$ and $G_2 \overset{rD}{\rightarrow} G_1$. (We call the first kind of relations symmetric and the latter two asymmetric.) E.g., $G_2 \overset{+}{\rightarrow} G_1$ is a shorthand for the combination of $G_2 \overset{+S}{\rightarrow} G_1$ and $G_2 \overset{+D}{\rightarrow} G_1$.

If $(G_1,...G_n) \xrightarrow{r}$ G is a goal relation we call $G_1,...G_n$ the source goals and G the target goal of *r*, and we say that *r* is an incoming relation for G and an outcoming relation for $G_1,...G_n$. Notice that all relations are directional, from the sources to the target goals. We call Boolean relations the *and* and *or* relations, partial contribution relations the + and - relations and their asymmetric versions, full contribution relations ++ and -- relations and their asymmetric versions. We call a root goal any goal with an incoming Boolean relation and no outcoming ones, we call a leaf goal any goal with no incoming Boolean relations.

We call a *path* from G_1 to G_k a sequence of goals $\pi := G_1,G_2\ldots,G_k$ s.t., for every $i \in \{1, ...,k-1\}$, G_i and G_{i+1} are respectively a source goal and the target goal of some relation r_i. We call a loop a path from a goal to itself. We call a diamond a pair of paths $\langle \pi_1, \pi_2 \rangle$ both from G_1 to G_k if π_1 and π_2 contain no common goal except G_1 and G_k. We call a goal graph a pair $\langle \mathbf{G},\mathbf{R} \rangle$ where **G** is a set of goal nodes and **R** is a set of goal relations, subject to the following restrictions:

each goal has at most one incoming Boolean relation; (1)

every loop contains at least one non-Boolean relation arc. (2)

In practice, a goal graph can be seen as a forest of AND/OR trees whose nodes are connected by contribution relationships. Root goals are roots of these trees, whilst leaf goals are either leaves or nodes that are not part of any tree.

The presence of contribution relations makes the tasks of formal reasoning on goal graphs less straightforward than in the case of simple AND/OR graphs. The following factors contribute to complicate the picture.

- **Asymmetric value propagation**. Satisfiability and deniability values may be propagated asymmetrically. For instance, the relation $G_2 \xrightarrow{++D} G_1$ suggests that the achievement of the goal G_2 is a necessary but not sufficient condition for achieving

goal G_1. In fact, if G_2 is denied, then there is full evidence that G_1 is denied, but if G_2 is satisfied, then nothing can be said about the satisfaction of G_1.

- **Partial evidence**. The contribution relations described above may propagate only partial evidence about the satisfiability/deniability of target goals. This means that a formal semantics for goal graphs must provide partial satisfiability/deniability values for the goals, as well as rules for propagating both full and partial satisfiability/deniability values through the relationships.

- **Conflicts**. Different goals can provide contradictory contributions to the same goals. For instance, if the graph contains $G_1 \overset{+S}{\rightarrow} G$ and $G_2 \overset{-S}{\rightarrow} G$ and both G_1 and G_2 are satisfied, then the first relation induces some evidence that G is satisfied, whilst the second induces some evidence that G is denied. We call these situations, conflicts. To this extent, it is important to keep track of both satisfiability and deniability values for all goals.

- **Diamonds**. The value of one goal alone can provide contradictory contributions to another goal due to the presence of diamonds. For instance, if the graph contains $(G_1, G_5) \overset{or}{\rightarrow} G_2$, $G_2 \overset{or}{\rightarrow} G_4$, $G_1 \overset{--S}{\rightarrow} G_3$ and $G_3 \overset{+D}{\rightarrow} G_4$, and both G_1 and G_5 are satisfied, then the satisfiability of G_1 propagates to G_4 through the diamond $\langle G_1G_2G_4, G_1G_3G_4\rangle$, providing both some evidence that G_4 is satisfied (path $G_1G_2G_4$) and some evidence that G_4 is denied (path $G_1G_3G_4$).

- **Loops**. The satisfiability/deniability of one goal can provide a contribution contradicting itself due to the presence of loops. This is the typical situation in

models containing negative feedback loops. For instance, if the graph contains $G_1 \xrightarrow{+} G_2$ and $G_2 \xrightarrow{-} G_1$, and if G_1 is satisfiable, then the fact that G_1 is satisfied propagates through G_2 providing some evidence that G_1 is denied.

2.2. Axiomatization of Goal Relationships

Let G_1, G_2, ... denote goals. We introduce four distinct predicates over goals, FS(G), FD(G) and PS(G), PD(G), meaning respectively that there is (at least) *full* evidence that goal G is satisfied and that G is denied, and that there is at least *partial* evidence that G is satisfied and that G is denied. We also use the proposition $\top$ to represent the (trivially true) statement that there is at least null evidence that the goal G is satisfied (or denied). Notice that the predicates state that there is at least a given level of evidence, because in a goal graph there may be multiple sources of evidence for the satisfaction/denial of a goal. We introduce a total order FS(G) ≥ PS(G) ≥ $\top$ and FD(G) ≥ PD(G) ≥ $\top$, with the intended meaning that $x \geq y$ if and only if $x \rightarrow y$. We call FS, PS, FD and PD the possible values for a goal.

We want to allow the deduction of positive ground assertions of type FS(G), FD(G), PS(G) and PD(G) over the goal constants of a goal graph. We refer to externally provided assertions as initial conditions. To formalize the propagation of satisfiability and deniability evidence through a goal graph $\langle \mathbf{G}, \mathbf{R} \rangle$, we introduce the axioms described in Figure 2. For instance, (3) states that full satisfiability and deniability imply partial satisfiability and deniability respectively; for an and relation, (4) states that full/partial satisfiability of a target node requires respectively full/partial satisfiability of all source nodes; for a "+S" relation, (8) states that only partial satisfiability (but not full satisfiability) propagates through a "+S" relation. Accordingly, an and relationship propagates the minimum satisfiability value (and the maximum deniability one), while a "+S" relation propagates at most a partial satisfiability value. To this extent, a "+S"

relation can be seen as an and relation with an unknown partially satisfiable goal. Similar considerations hold for the other relationships.

Goal	Invariant Axioms	
G:	$\mathsf{FS}(G) \rightarrow \mathsf{PS}(G), \quad \mathsf{FD}(G) \rightarrow \mathsf{PD}(G)$	(3)
Goal relation	**Relation Axioms**	
$(G_1,...,G_i,...G_n) \overset{and}{\longmapsto} G$:	$(\bigwedge_i \mathsf{FS}(G_i)) \rightarrow \mathsf{FS}(G), \ (\bigwedge_i \mathsf{PS}(G_i)) \rightarrow \mathsf{PS}(G)$	(4)
	$\bigwedge_i (\mathsf{FD}(G_i) \rightarrow \mathsf{FD}(G)), \ \bigwedge_i (\mathsf{PD}(G_i) \rightarrow \mathsf{PD}(G))$	(5)
$(G_1,...,G_i,...G_n) \overset{or}{\longmapsto} G$:	$(\bigwedge_i \mathsf{FD}(G_i)) \rightarrow \mathsf{FD}(G), \ (\bigwedge_i \mathsf{PD}(G_i)) \rightarrow \mathsf{PD}(G)$	(6)
	$\bigwedge_i (\mathsf{FS}(G_i) \rightarrow \mathsf{FS}(G)), \ \bigwedge_i (\mathsf{PS}(G_i) \rightarrow \mathsf{PS}(G))$	(7)
$G_2 \overset{+S}{\longmapsto} G_1$:	$\mathsf{PS}(G_2) \rightarrow \mathsf{PS}(G_1)$	(8)
$G_2 \overset{-S}{\longmapsto} G_1$:	$\mathsf{PS}(G_2) \rightarrow \mathsf{PD}(G_1)$	(9)
$G_2 \overset{++S}{\longmapsto} G_1$:	$\mathsf{FS}(G_2) \rightarrow \mathsf{FS}(G_1), \ \mathsf{PS}(G_2) \rightarrow \mathsf{PS}(G_1)$	(10)
$G_2 \overset{--S}{\longmapsto} G_1$:	$\mathsf{FS}(G_2) \rightarrow \mathsf{FD}(G_1), \ \mathsf{PS}(G_2) \rightarrow \mathsf{PD}(G_1)$	(11)
$G_2 \overset{+D}{\longmapsto} G_1$:	$\mathsf{PD}(G_2) \rightarrow \mathsf{PD}(G_1)$	(12)
$G_2 \overset{-D}{\longmapsto} G_1$:	$\mathsf{PD}(G_2) \rightarrow \mathsf{PS}(G_1)$	(13)
$G_2 \overset{++D}{\longmapsto} G_1$:	$\mathsf{FD}(G_2) \rightarrow \mathsf{FD}(G_1), \ \mathsf{PD}(G_2) \rightarrow \mathsf{PD}(G_1)$	(14)
$G_2 \overset{--D}{\longmapsto} G_1$:	$\mathsf{FD}(G_2) \rightarrow \mathsf{FS}(G_1), \ \mathsf{PD}(G_2) \rightarrow \mathsf{PS}(G_1)$	(15)

Figure 2. Ground axioms for the invariants and the propagation rules

Notice that, combining (3) with (4), and (3) with (8), we have, respectively,

$$(G_2,G_3) \overset{and}{\rightarrow} G_1\text{: } (\mathrm{FS}(G_2) \wedge \mathrm{PS}(G_3)) \rightarrow \mathrm{PS}(G_1) \tag{16}$$

$$G_2 \overset{+S}{\rightarrow} G_1\text{: } \mathrm{FS}(G_2) \rightarrow \mathrm{PS}(G_1) \tag{17}$$

To this extent, henceforth we implicitly assume that axioms (3) are always implicitly applied whenever possible. Thus, e.g., we say that PS(G_1) is deduced from FS(G_2) and PS(G_3) by applying (4) —meaning "applying (3) and then (4)" — or that PS(G_1) is deduced from FS(G_2) and FS(G_3) by applying (4) —meaning "applying (4) and then (3)".

Let $\mathbf{A}\text{: } (\wedge_{i=1}^{n} v_i) \rightarrow v$ be a generic relationship axiom for the relation r. We call the values v_i prerequisite values, and v the consequence value of axiom **A**, and we say that the

values v_i are the prerequisites for v through r and that v is the consequence of the values v_i through r. We say that an atomic proposition of the form FS(G), FD(G), PS(G) and PD(G) holds if either it is an initial condition or it can be deduced via modus ponens from the initial conditions and the ground axioms of Figure 2. We assume conventionally that T always holds. Notice that all the formulas in the framework described so far are propositional Horn clauses, so that deciding if a ground assertion holds not only is decidable, but also it can be decided in polynomial time.

A *weak conflict* holds if (PS(G) ∧ PD(G)), a medium conflict holds if either (FS(G) ∧ PD(G)) or (PS(G) ∧ FD(G)), while a strong conflict holds if (FS(G) ∧ FD(G)), for some goal G.

3. Forward Reasoning

Based on the framework introduced in Section 2, [9, 10] present algorithms for propagating through a goal graph $\langle \mathbf{G}, \mathbf{R} \rangle$ labels representing evidence for the satisfiability and deniability of goals. To each node $G \in \boldsymbol{G}$ we associate two variables $Sat(G)$ and $Den(G)$ ranging within $\{F, P, N\}$ (full, partial, none) such that $F > P > N$, representing the current evidence of satisfiability and deniability of goal G. For example, $Sat(G_i) \geq P$ states that there is at least partial evidence that Gi is satisfiable. Starting from assigning an initial set of input values for $Sat(G_i)$, $Den(G_i)$ to (a subset of) the goals in G, we propagate the values through the goal relations in $\boldsymbol{R}$ according to the propagation rules of Table 1. The schema of the algorithm is described in Figure 3. *Initial*, *Current* and *Old* are arrays of $|\boldsymbol{G}|$ pairs $\langle Sat(G_i), Den(G_i) \rangle$, one for each $G_i \in \boldsymbol{G}$, representing respectively the initial, current and previous labeling states of the graph. We call the pair $\langle Sat(G_i), Den(G_i) \rangle$ a *label* for G_i. Notationally, if W is an array of labels

$\langle Sat(G_i), Den(G_i)\rangle$, by $W[i].sat$ and $W[i].den$ we denote the first and second field of the ith label of W.

The array *Current* is first initialized to the initial values *Initial* given as input by the user. At each step, for every goal G_i, $\langle Sat(G_i), Den(G_i)\rangle$ is updated by propagating the values of the previous step. This is done until a fixpoint is reached, in the sense that no further updating is possible (*Current* == *Old*).

The updating of $\langle Sat(G_i), Den(G_i)\rangle$ works as follows. For each relation R_j incoming in G_i, the satisfiability and deniability values sat_{ij} and den_{ij} derived from the old values of the source goals are computed by applying the rules of Table 1. The result is compared with the old value, and the maximum is returned as new value for G_i.

In [9, 10] we have showed that the algorithm is sound and complete with respect to the axiomatization.

	$(G_2, G_3) \overset{and}{\longmapsto} G_1$	$G_2 \overset{+S}{\longmapsto} G_1$	$G_2 \overset{-S}{\longmapsto} G_1$	$G_2 \overset{++S}{\longmapsto} G_1$	$G_2 \overset{--S}{\longmapsto} G_1$
$Sat(G_1)$	$min\left\{\begin{matrix} Sat(G_2), \\ Sat(G_3) \end{matrix}\right\}$	$min\left\{\begin{matrix} Sat(G_2), \\ P \end{matrix}\right\}$	N	$Sat(G_2)$	N
$Den(G_1)$	$max\left\{\begin{matrix} Den(G_2), \\ Den(G_3) \end{matrix}\right\}$	N	$min\left\{\begin{matrix} Sat(G_2), \\ P \end{matrix}\right\}$	N	$Sat(G_2)$
	$(G_2, G_3) \overset{or}{\longmapsto} G_1$	$G_2 \overset{+D}{\longmapsto} G_1$	$G_2 \overset{-D}{\longmapsto} G_1$	$G_2 \overset{++D}{\longmapsto} G_1$	$G_2 \overset{--D}{\longmapsto} G_1$
$Sat(G_1)$	$max\left\{\begin{matrix} Sat(G_2), \\ Sat(G_3) \end{matrix}\right\}$	N	$min\left\{\begin{matrix} Den(G_2), \\ P \end{matrix}\right\}$	N	$Den(G_2)$
$Den(G_1)$	$min\left\{\begin{matrix} Den(G_2), \\ Den(G_3) \end{matrix}\right\}$	$min\left\{\begin{matrix} Den(G_2), \\ P \end{matrix}\right\}$	N	$Den(G_2)$	N

Table 1. Propagation rules in the qualitative framework.

```
1   label_array Label_Graph(graph⟨G,R⟩, label_array Initial)
2       Current=Initial
3       do
4           Old=Current
5           for each Gi∈G do
6               Current[i]=Update_label(i,⟨G,R⟩,Old);
```

```
7         until (Current==Old);
8         return Current;
9
10   label Update_label(int i, graph⟨G,R⟩, label_array Old)
11        for each R_j ∈ R s.t. target(R_j)==G_i do
12             sat_ij = Apply_Rules_sat(G_i,R_j,old);
13             den_ij = Apply_rules_Den(G_i,R_j,Old);
14        return ⟨ max(max_j(sat_ij),Old[i].sat),
                   max(max_j(den_ij),Old[i].den) ⟩
```

Figure 3. Schema for the label propagation algorithm.

4. Backward Reasoning

In this section, we focus on the backward search of the possible input values leading to some desired final value, under desired constraints. The user sets the desired final values of the target goals, and the system looks for possible initial assignments to the input goals which would cause the desired final values of the target goals by forward propagation. The user may also add some desired constraints, and decide to avoid strong/medium/weak confiicts. As we said in the introduction the problems related to the backward reasoning are solved respectively by reducing them into those of satisfiability and minimum weight satisfiability of Boolean formulas. To be self-contained, before going in the details of backward reasoning, we recall briefly some basic notions about Boolean satisfiability and minimum-weight Boolean satisfiability.

4.1 SAT and Minimum-Cost SAT

Propositional satisfiability (SAT) is the problem of determining whether a Boolean formula Φ admits at least one satisfying truth assignment μ to its variables A_i. In a broad sense, a SAT solver is any procedure that is able to decide such a problem. SAT is an NP-complete problem [4], so that we can reasonably assume that there does not exist any polynomial algorithm that solves it.

Recent years have witnessed an impressive advance in the efficiency of SAT techniques, which has brought large previously intractable problems at the reach of state-of-the-art solvers (see [20] for an overview).

The most popular SAT algorithm is DPLL [7] in its many variants, and Chaff [12] is probably the most efficient DPLL implementation available. In its basic version, DPLL tries to find a satisfying assignment recursively by assigning, at each step, a value to a proposition. The input formula must be previously reduced in conjunctive normal form (CNF)[1]. At each step, if there exists a unit clause, DPLL assigns it to true; otherwise, it chooses a literal l and it tries to find an assignment with l set to true; if it doesn't succeeds, it tries with l set to false. In this way, DPLL performs the deterministic choices first while postponing, as far as possible the branching step, which is the main source of exponential blow up. There are several techniques to improve the efficiency of DPLL such as, e.g., back-jumping, learning, random restart (again, see [20] for an overview).

A noteworthy variant of SAT is Minimum-Weight Propositional Satisfiability (hereafter MW-SAT) [12]. The Boolean variables Ai occurring in Φ are given a positive integer weight wi, and MW-SAT is the problem of determining a truth assignment μ satisfying Φ which minimizes the value

$$W(\mu) := \sum_i \{w_i \mid A_i \text{ is assigned } \mathsf{T} \text{ by } \mu\} \tag{18}$$

or stating there is none. In the general case MW-SAT is Δ_2^p-complete problem[2] [12]. That is, it is much harder than simple SAT. The state-of-the-art solver for MW-SAT is Minweight [12], based on a variant of the DPLL procedure.

4.2 Input and Target Goals

The notions of “input goal" and “target goal" deserve some more comments. Goal graphs may contain cycles so that, in principle, it is not obvious a priori which goals are target/output goals and which are input ones. Although in our experience the Boolean relations tend to have a dominant role, so that target goals are typically roots and input goals are typically leaves, the choice is typically left to the user.

Nevertheless, the choice is not completely free, as we impose that every path incoming in a target goal must be originated in an input node, that is:

for every target goal G there exists a direct acyclic subgraph (DAG) rooted in G whose leaves $G_i \ldots G_{ik}$ are input nodes, (19)

so that the value of G derives by forward propagation from those of $G_i \ldots G_{ik}$. An easy-to-verify sufficient condition for (19) is that

all leaf goals are input goals[3] (20)

Example 1. Consider the simple goal graph of Figure 4, and suppose that G_0 is the target goal and G_2 and G_3 are the input goals. (Notice that G_0 and G_1 form a loop without input goals.) If we assigned a final value $FS(G_0)$, then by backward search we could have $FS(G_1)$ and then $FS(G_0)$ again. Thus, $FS(G_0)$ could be derived by forward propagation from itself without any input value, which is a nonsense. If instead G1 is an input goal, then by backward search weobtain $FS(G_1)$ or $FS(G_2)$ and $FS(G_3)$, which are suitable initial assignments to the input goals. Notice that

$(G_2;G_3) \overset{and}{\rightarrow} G_0$ and $G_1 \overset{++S}{\rightarrow} G_0$ form a DAG rooted in G_0 whose leaves are input nodes.

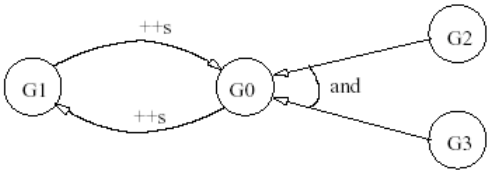

Figure 4. The goal graph of Example 1.

4.3 Basic Formalization

We want to reduce the problem of backward search for input values to that of satisfiability (SAT) of a Boolean formula Φ. The Boolean variables of Φ are all the values FS(G), PS(G), FD(G), PD(G) for every goal G $\in$ **G**, and Φ is written in the form:

$$\Phi := \Phi_{graph} \wedge \Phi_{outval} \wedge \Phi_{backward} \left[\wedge \Phi_{constra\,int\,s} \wedge \Phi_{conflict} \right] \tag{21}$$

where the conjuncts Φ_{graph}, Φ_{outval}, Φ_{backward} and optional components $\Phi_{\text{constraints}}$, and Φ_{conflict} are explained below.

Encoding the Goal Graph: Φ_{graph}. The first component Φ_{graph} is the representation of the goal graph $\langle \mathbf{G}, \mathbf{R} \rangle$, given in the form

$$\Phi_{graph} := \bigwedge_{G \in \mathbf{G}} In\,var\_Ax(G) \wedge \bigwedge_{r \in \mathbf{R}} Rel\_Ax(r) \tag{22}$$

Invar_Ax(G) being the conjunction of the invariant axioms (1) and (2) for the goal G in Figure 2 and *Rel_Ax(r)* being the conjunction of the relation axioms in (3)-(12) and their dual ones corresponding to the relation *r*. These axioms encode the forward propagation of values through the relation arcs in the goal graph.

Representing Desired Final Output Values: Φ_{outval}. The second component Φ_{outval} is a representation of the output values the user want to be assigned to the target goal. Φ_{outval} is written in the form:

$$\Phi_{outval} := \bigwedge_{G \in Target(\mathbf{G})} vs(G) \wedge \bigwedge_{G \in Target(\mathbf{G})} vd(G) \tag{23}$$

Target(G) being the set of target goals in **G** and *vs(G)* ∈ {T; PS(G); FS(G)}, vd(G) ∈ {T; PD(G); FD(G)} being the maximum satisfiability and deniability values assigned by the user to the target goal G. Φ_{outval} is a conjunction of unit clauses, which force the desired output values vs(G) and vd(G) to be assigned to ⊤.

Encoding Backward Reasoning: Φbackward. The third component Φ_{backward} encodes the backward search. Φ_{backward} is written in the form:

$$\Phi_{backward} := \bigwedge_{G \in \mathbf{G} / Input(\mathbf{G})} \bigwedge_{v(G)} Backward\_Ax(v(G)) \tag{24}$$

$$Backward\_Ax(v(G)) := v(G) \rightarrow \bigvee_{r \in Incoming(G)} Prereqs(v(G), r) \tag{25}$$

*Input(**G**)* being the set of input goals in **G**, *Incoming(**G**)* being the set of relations incoming in **G**, *v(G)* ∈ {PS(G); FS(G); PD(G); FD(G)}, and *Prereqs(v(G); r)* is a formula which is true if and only if the prerequisites of v(G) through *r* hold. The list of possible backward propagation axioms *Backward_Ax(v(G))* is reported in Figure 5, (26)-(29).

Suppose *G* is not an input goal. If *v(G)* holds, then this value must derive from the prerequisite values of some of the incoming relations of *G*. *Prereqs(v(G); r)* are exactly

the conditions which must be verified to apply the corresponding relation axioms (3)-(12) and their dual ones in Figure 2.

$$
\begin{array}{lll}
\textbf{Goal} & \textbf{Backward Propagation Axioms} & \\
G\ (non-input): & FS(G) \rightarrow \begin{pmatrix}
\bigwedge_i FS(G_i) \vee & \text{If } (G_1,...,G_i,...G_n) \overset{and}{\longmapsto} G \\
\bigvee_i FS(G_i) \vee & \text{If } (G_1,...,G_i,...G_n) \overset{or}{\longmapsto} G \\
FS(G_i) \vee & \text{For every relation } R_i\text{: } G_i \overset{++_S}{\longmapsto} G \\
FD(G_i) & \text{For every relation } R_i\text{: } G_i \overset{--_D}{\longmapsto} G
\end{pmatrix} & (26) \\
& PS(G) \rightarrow \begin{pmatrix}
\bigwedge_i PS(G_i) \vee & \text{If } (G_1,...,G_i,...G_n) \overset{and}{\longmapsto} G \\
\bigvee_i PS(G_i) \vee & \text{If } (G_1,...,G_i,...G_n) \overset{or}{\longmapsto} G \\
PS(G_i) \vee & \text{For every relation } R_i\text{: } G_i \overset{++_S}{\longmapsto} G \\
PD(G_i) \vee & \text{For every relation } R_i\text{: } G_i \overset{--_D}{\longmapsto} G \\
PS(G_i) \vee & \text{For every relation } R_i\text{: } G_i \overset{+_S}{\longmapsto} G \\
PD(G_i) & \text{For every relation } R_i\text{: } G_i \overset{-_D}{\longmapsto} G
\end{pmatrix} & (27) \\
& FD(G) \rightarrow \begin{pmatrix}
\bigvee_i FD(G_i) \vee & \text{If } (G_1,...,G_i,...G_n) \overset{and}{\longmapsto} G \\
\bigwedge_i FD(G_i) \vee & \text{If } (G_1,...,G_i,...G_n) \overset{or}{\longmapsto} G \\
FD(G_i) \vee & \text{For every relation } R_i\text{: } G_i \overset{++_D}{\longmapsto} G \\
FS(G_i) & \text{For every relation } R_i\text{: } G_i \overset{--_S}{\longmapsto} G
\end{pmatrix} & (28) \\
& PD(G) \rightarrow \begin{pmatrix}
\bigvee_i PD(G_i) \vee & \text{If } (G_1,...,G_i,...G_n) \overset{and}{\longmapsto} G \\
\bigwedge_i PD(G_i) \vee & \text{If } (G_1,...,G_i,...G_n) \overset{or}{\longmapsto} G \\
PD(G_i) \vee & \text{For every relation } R_i\text{: } G_i \overset{++_D}{\longmapsto} G \\
PS(G_i) \vee & \text{For every relation } R_i\text{: } G_i \overset{--_S}{\longmapsto} G \\
PD(G_i) \vee & \text{For every relation } R_i\text{: } G_i \overset{+_D}{\longmapsto} G \\
PS(G_i) & \text{For every relation } R_i\text{: } G_i \overset{-_S}{\longmapsto} G
\end{pmatrix} & (29)
\end{array}
$$

Figure 5. Axioms for backward propagation

Adding User's Constraints and Desiderata The first optional component *Φconstraints* allows the user to express constraint and desiderata on goal values. *Φ*constraints is generically written in the form:

$$\Phi_{outval} := \bigvee_i \bigwedge_j lit_{ij} \tag{30}$$

litij ∈ {PS(G); FS(G); PD(G); FD(G); ¬PS(G); ¬FS(G); ¬PD(G); ¬FD(G)}, G ∈ **G**. A positive unit clause value is used to impose a minimum value to a goal. (E.g., "$PS(G_1)$"

means “G_1 is at least partially satisfiable", but it might be totally satisfiable.) A negative unit clause value is used to prevent a value to a goal. (E.g., “$FD(G_1)$" means “G_1 cannot be fully deniable", but it might be partially deniable.) A disjunction of positive values is used to state an alternative desideratum. (E.g., “$FS(G_1) \vee FS(G_2)$" means “at least one between G_1 and G_2 must be fully satisfiable".) A disjunction of negative values is used to state a mutual exclusion constraint. (E.g., “$FD(G_1) \vee FD(G_2)$” means “G_1 and G_2 cannot be both fully deniable”, but they can be partially deniable.)

Preventing conflicts. The second optional component Φ_{conflict} allows the user for looking for solutions which do not involve conflicts. Depending whether one wants to avoid (i) only the strong conflicts, (ii) the strong and medium conflicts or (iii) all conflicts, Φ_{conflict} is encoded respectively as follows:

$$\Phi_{conflict} := \bigwedge_{G \in \mathbf{G}} (\neg FS(G) \vee \neg FD(G)) \tag{31}$$

$$\Phi_{conflict} := \bigwedge_{G \in \mathbf{G}} (\neg FS(G) \vee \neg PD(G)) \wedge (\neg PS(G) \vee \neg FD(G)) \tag{32}$$

$$\Phi_{conflict} := \bigwedge_{G \in \mathbf{G}} (\neg PS(G) \vee \neg PD(G)) \tag{33}$$

(31) states that G cannot be fully satisfiable and fully deniable; (32) states that G cannot be fully satisfiable and (fully or) partially deniable, and vice versa; (33) states that G cannot be (fully or) partially satisfiable and (fully or) partially deniable. Notice that, by Axioms (1)-(2), (33) implies (32) and that (32) implies (29).

4.4 Solving Simple and Minimum-Cost Goal Satisfiability

Consider a goal graph $\langle \mathbf{G}, \mathbf{R} \rangle$ with input goals $G_{i1} \ldots G_{ik}$ and target goals $G_{f1} \ldots G_{fn}$, and a set of desired final values $vs(G_{f1})$, $vd(G_{f1})$, ... $vs(G_{fn})$, $vd(G_{fn})$ to the target goals (plus possibly a set of user constraints and desiderata), and let Φ be the formula encoding the problem, as in (18).

The correctness and completeness of the whole approach has been proved in [18]. According to the proof, (i) if Φ is unsatisfiable, then no value exists to the input goals from which the desired final values derive by forward propagation (verifying the desiderata and constraints) (ii) if an assignment μ satisfying Φ exists, then the maximum values $vs(G_{i1}), vd(G_{i1}), \ldots vs(G_{in}), vd(G_{ik})$ which μ assigns to T are such that the desired final values derive from them by forward propagation (verifying the desiderata and constraints). This allows us to reduce the problem of backward search to that of propositional satisfiability.

Goalsolve

We have implemented an algorithm, called Goalsolve, for the backward search of the possible input values leading to some desired final value, under desired constraints. The schema of Goalsolve is reported in Figure 6 (black arrows).

Goalsolve takes as input a representation the goal graph, a list of desired final values and, optionally, a list of user desiderata and constraint and a list of goals which have to be considered as input. (The default choice is that indicated in condition (20), that is, all leaf goals are considered input goals.) The user may also activate some flags for switching on the various levels of “avoiding conflicts".

The first component of Goalsolve is an encoder that generates the Boolean CNF formula Φ as described in previous section, plus a correspondence table Table between goal values and their correspondent Boolean variable. Φ is given as input to the SAT solver Chaff [13], which returns either “UNSAT" if Φ is unsatisfiable, or “SAT" plus a satisfying assignment μ if Φ is satisfiable. Then a decoder uses Table to decode back the resulting assignment into the set of goal values.

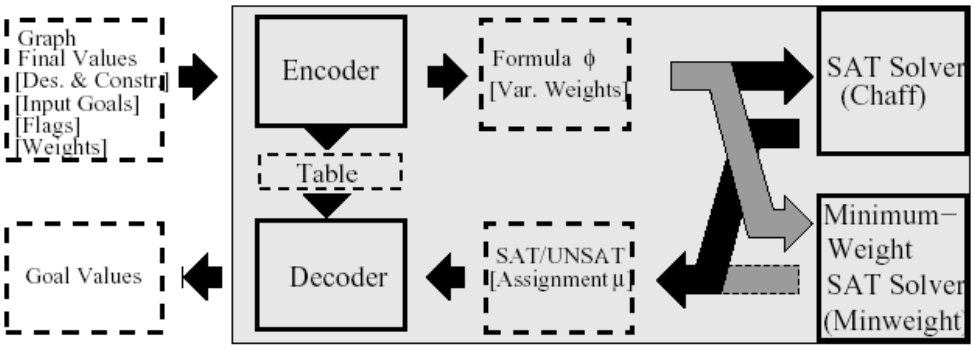

Figure 6. Schema of Goalsolve (black arrows) and Goalminsolve (gray arrows).

Goalminsolve

In general, the satisfaction/deniability value of (input) goals may have different costs. Thus we have implemented a variant of Goalsolve, called Goalminsolve, for the search of the goal values of minimum cost. The schema of Goalminsolve is reported in Figure 6 (gray arrows). Unlike Goalsolve, Goalminsolve takes as input also a list of integer weights W(val(G)) for the goal values. (The default choice is W(FS(G)) = (FD(G)) = 2 and W(PS(G)) = (PD(G)) = 1 if G is an input goal, W(FS(G)) = (FD(G)) = W(PS(G)) = (PD(G)) = 0 otherwise.) The encoder here encodes also the input weight list into a list of weights for the corresponding Boolean variables of Φ. Both Φ and the list of weights are given as input to the minimum-weight SAT solver Minweight [12], which returns either "UNSAT" if Φ is unsatisfiable, or "SAT" plus a minimum-weight satisfying assignment μ if Φ is satisfiable. The decoder then works as in Goalsolve.

Notice that, in general, there may be plenty many satisfying assignments -- up to exponentially many-- corresponding so solutions for the problem. In a typical session with Goalsolve or Goalminsolve, the user may want to work first with the "avoiding conflicts" flags, starting from the most restrictive (33) down to the least restrictive (31), until the problem admits solution. (E.g., it is often the case that no solution avoiding all conflicts exists, but if one allows for weak AND/OR medium conflicts a solution

exists.) Then, once the level of conflict avoidance is fixed, the user may want to work on refining the solution obtained, by iteratively adding positive and negative values -- e.g. "FD(G_1)", "FS(G_2)" -- in the list of desiderata and constraints, until a satisfactory solution is found.

5. Using Goal Analysis

We briefly describe in this section how goal analysis and reasoning mechanisms are used within the Tropos methodology. In particular, we focus on the Tropos requirements phases and we illustrate how forward and backward reasoning are used in the Media@ case study[4].

Let's consider the actor diagram presented in Figure 7. The figure shows part of the goal analysis for the Media@ system (the system-to-be) related to the goal *manage internet shop*, as presented in [2]. The goal is firstly refined into goals *manage internet order*, *manage item searching*, *produce statistics* and *adaptation*. To achieve *manage internet order* is used the goal *shopping cart* which is decomposed into subgoals *select item*, *add item*, *check out,* and *get identification details*. These are the main process activities required to design an operational on-line shopping cart [5]. The latter (goal) is achieved either through subgoal *classic communication handled* dealing with phone and fax orders or *internet handled* managing secure or standard form orderings. To allow for the ordering of new items not listed in the catalogue, *select item* is also further refined into two alternative subgoals, one dedicated to select catalogued items, the other to pre-order unavailable products. To provide sufficient support (++) to the *adaptability* softgoal, *adaptation* is refined into four subgoals dealing with catalogue updates, system evolution, interface updates and system monitoring. The goal *manage item searching* might alternatively be fulfilled through goals *DB querying* or *catalogue*

consulting with respect to customers' navigating desiderata, i.e., searching with particular items in mind by using search functions or simply browsing the catalogued products.

Figure 7 reports also the analysis for the softgoals *security* and *usability*. *Security* receives positive contribution from the satisfaction of softgoals *privacy*, *availability*, and *integrity*, whereas *usability* from *adaptability* and *easy to use*. Notice, that *standard form order* gives a negative contribution to the *privacy*. Of course the analysis should include other non-function requirements, but for sake of simplicity we just focus on these two.

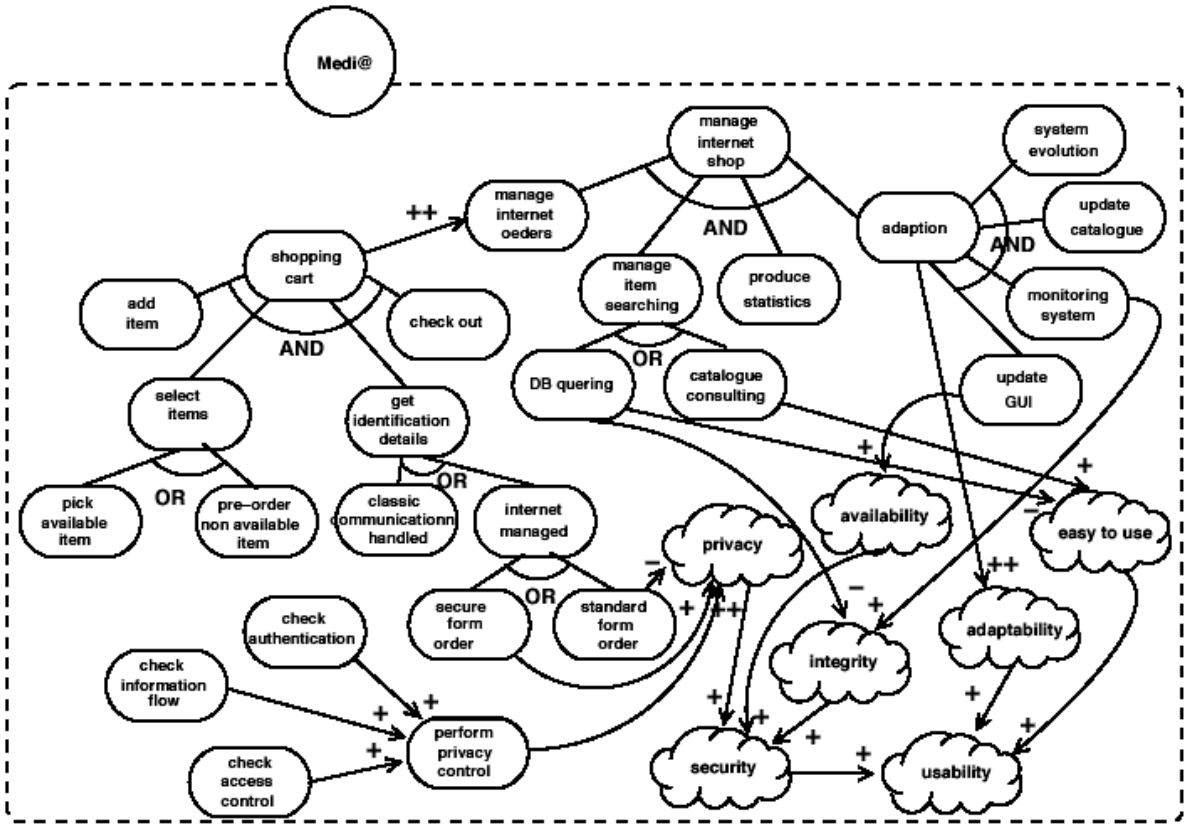

Figure 7. Actor diagram for the Media Shop focusing on the goal *internet shop managed*

5.1 Using Forward Reasoning

Forward reasoning is adopted in Tropos in order to evaluate the impact of the adoption of the different alternatives with respect to the softgoals of the system-to-be. Table 2 reports the results of the forward reasoning in four different situations for the goal model presented in Figure 7. The table shows only the results for the goals involved in OR decompositions, the top goal *manage internet shop*, and all the softgoals of the model. For all the other (leaf) goals we assume they have full evidence for satisfaction as initial assignment. For each experiment, the table reports the initial (Init) and final (Fin) values assumed by each goal.

In the first experiment (Exp1) we adopt the goal *DB querying* as the choice to achieve *manage item searching*, the goal *pick available item* to achieve *select items*, and the goal *classic communication handled* to achieve *get identification details*. The result is that the top goal *manage internet shop* is fully satisfied (Sat(...)=F) and all the softgoals are at least partial satisfied (Sat(...)=P), except the softgoal *easy to use* that results partially denied (Den(...)=P). Notice also that this initial assignment produces a conflict for the *integrity* softgoal (Sat(...)=P and Den(...)=P). In the second experiment (Exp2) we adopt the goal *standard form order* instead of the goal *classic communication handled*. This mainly produce the result of moving the conflict from the softgoal *integrity* to the softgoal *privacy*. In the third experiment (Exp3) we decide to *manage item searching* using the *catalogue consulting* goal. The effect of this new assignment is that softgoal *easy to use* is now partially satisfied, but we have conflicts for softgoals *integrity* and *privacy*. Finally, in the fourth experiment (Exp4) we adopt *secure form order* instead of the *standard form order* goal. This has the effect that now we do not have conflicts and all the softgoals are at least partially satisfied.

Goals	Exp 1				Exp 2				Exp 3				Exp 4			
	Init		Fin		Init		Fin		Init		Fin		Init		Fin	
	S	D	S	D	S	D	S	D	S	D	S	D	S	D	S	D
DB querying	F		F		F		F									
catalogue consulting									F		F		F		F	
pick available item	F		F		F		F		F		F		F		F	
pre-order non available item																
classic communication handled	F		F													
standard form order					F		F		F		F					
secure form order													F		F	
manage internet shop			F				F				F				F	
privacy			P				P	P			P	P			P	
availability			P				P				P				P	
integrity			P	P			P				P	P			P	
usability			P				P				P				P	
adaptability			F				F				F				F	
easy to use				P				P			P				P	
security			P				P				P				P	

Table 2. Evaluating alternatives in the goal model of Figure 7.

Table 2 reports results limited to the simple model of Figure 7. Also in the model we have used only symmetric relationships and we have not distinguished between relations *+S* and *+D* or *-S* and *-D*. In real-life case studies, the goal models to be analyzed are usually more complex. For instance, in [15], we have presented a goal model with more than hundred goals for the Trentino Public Transportation System, in which non symmetric relationships have been used.

The analysis presented above concerns only a goal model and does not consider the effects of a particular assignment to the goals of other goal models. This kind of analysis is called *intra actor* analysis since it does not involve goal models of other actors. Differently, the *inter actor* analysis extends the boundary of the analysis to the goal models of the other actors. So for instance, we could analyze the effects of an assignment of Table 2 to the softgoals like *happy customers* and *improve quality of services* that are part of the goal models of other actors.

5.2 Using Backward Reasoning

Backward reasoning is used to discover solutions or minimal solutions for the fulfilment of root goals. Table 3 presents the results of backward reasoning in four different situations with the goal model of the *Medi@* shop presented in Figure 7. The cost of each alternative goal is reported near its label (e.g., the cost of the *DB querying* is 3.). In the first experiment we try to find an assignment at the minimal cost that allows to obtain the full satisfaction of the top goal *manage internet shop* and all the softgoals. Unfortunately, no solution exists and this is due to the fact that almost all the softgoals receive only (+) and no (++) contributions. In the second experiment we require the full satisfaction of the top goal *manage internet shop* and partial satisfaction of all the softgoals. The solution at the minimum cost results the full satisfaction for *catalogue consulting*, *pick available item* and *secure form order*. In the third experiments, we relaxed the constraint of avoiding conflicts and we obtain that now the solution includes the full satisfaction of the goal *standard form order* instead of the goal *secure form order*. Of course, now in the final values of the target goals we have conflicts, and in particular a conflict for the goal *privacy* (Sat(...)=P and Den(...)=P) and the goal *integrity* (Sat(...)=P and Den(...)=P). In the final experiment we imposed only the full satisfaction for the goal *manage internet shop* and the softgoal *privacy*. The solution with no conflicts is reported in table.

Also for backward reasoning the analysis can be extended to the goal models outside the boundary of the single actor. In this case the desired values can be assigned to (soft)goals of different goal models and the final solution will include goals of one or more goal models.

Goals	Exp 1				Exp 2				Exp 3				Exp 4			
	Init		Fin		Init		Fin		Init		Fin		Init		Fin	
	S	D	S	D	S	D	S	D	S	D	S	D	S	D	S	D
DB querying (3)															F	
catalogue consulting (6)							F				F					
pick available item (2)							F				F				F	
pre-order non available item (7)																
classic communication handled (4)																
standard form order (6)											F					
secure form order (8)							F								F	
manage internet shop	F				F		F		F		F		F		F	
privacy	F				P		P		P		P	P	P		P	
availability	F				P		P		P		P				P	
integrity	F				P		P		P		P	P			P	
usability	F				P		P		P		P				P	
adaptability	F				P		F		P		F				F	
easy to use	F				P		P		P		P					P
security	F				P		P		P		P				P	

Table 3. Backward reasoning with the goal model of Figure 7.

6. The Goal Reasoning Tool

Forward and backward reasoning are both supported in Tropos by the Goal Reasoning Tool (GR-Tool). This is a CASE tool developed for modelling and verifying requirements models by supporting goal analysis. Specifically, the GR-Tool offers a *Graphical environment, a Goal reasoning interface,* and a *Translation tool*.

The **Graphical environment** is a visual framework for creating and manipulating goal diagrams. The user can choose to draw Tropos diagrams (actor and goal diagrams), or a standalone goal diagram that is not associated to any actor. In both cases the user is requested to specify details about each single goal, such as whether the goal is an input /root goal, as well as values for the variables SAT and DEN. Figure 8 shows an example of Tropos diagram within the environment. In the left-hand side of the interface the user can specify goal details.

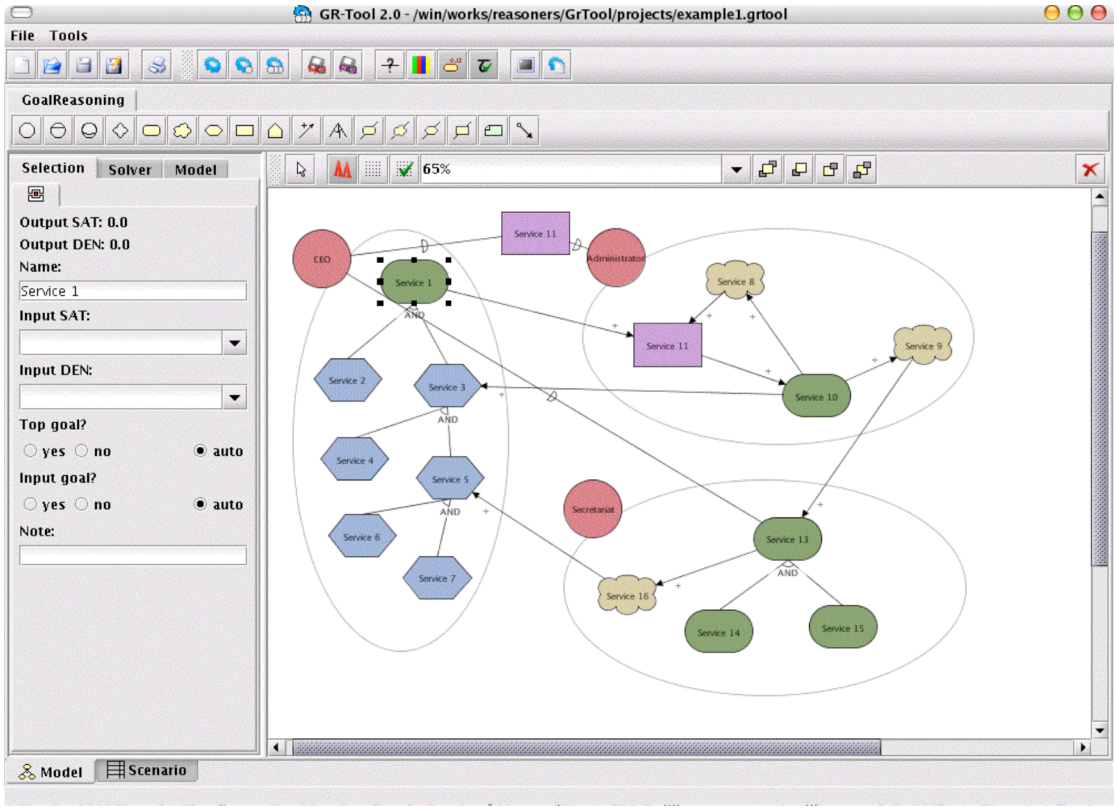

Figure 8. A snapshot of the GR-Tool

The **Goal Reasoning interface** is a front-end to external tools for forward and backward reasoning. The user can decide to execute four different type of analysis: *qualitative forward-analysis*, *qualitative backward-analysis*, *quantitative forward-analysis*, and *quantitative backward-analysis*. It is possible to move from one type of analysis to another one without the need to re-input details on the analysis (input/target goals, desired/input values). Qualitative values are transformed in quantitative and vice versa (FS->S=1.0; PS->0.5; N->0; etc.). Figure 9 shows the panel where to input some specific parameters of the quantitative backward analysis.

Goal analyses can be conducted for different types of diagrams:

• *Single actor diagrams*, where the analysis is done on a single actor's goal diagram;

• *Multiple actor diagram*s, where the analysis is done on a diagram generated by joining the goal diagrams of several actors;

• *Selection diagrams*, where the analysis is done on a diagram created joining pieces of goal diagrams.

For each type of analysis we can use different scenarios to compare results obtained with different input values (e.g., varying the desired final values for the target goals, avoiding conflicts, with weak conflicts, etc.). Results can be shown graphically (directly on the diagram) or in tabular way (showing the values for each goal).

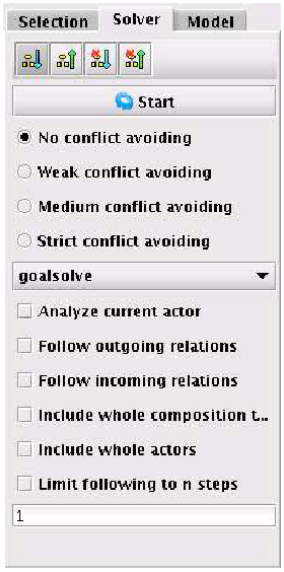

Figure 9. Quantitative backward analysis options

The **Translation tool** allows the user to export, and import Tropos diagram in Formal Tropos specification language, XML, or Datalog. This is very useful for the integration of the GR-Tool functionalities and the functionalities of other Tropos tools, such as for example ST-Tool and T-Tool. Figure 10 shows an example of specification in Formal Tropos.

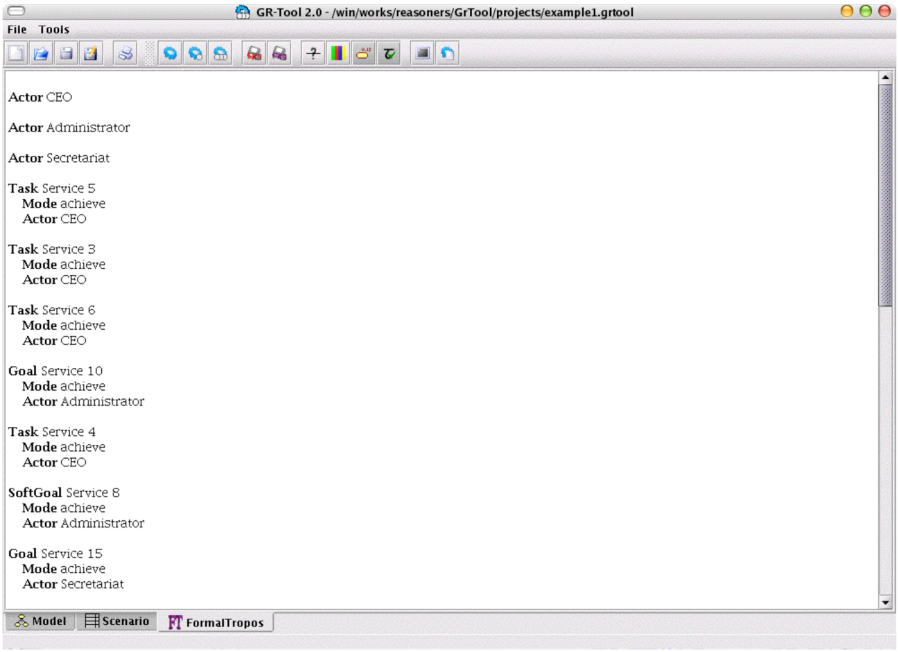

Figure 10. Formal Tropos specification

The GR-Tool consists of a GR-Tool kernel and external solvers. GR-Tool (version 2.0) is a cross-platform swing-based Java application. It requires a Java virtual machine - v. 1.4.1 or above - properly installed to run. The package also includes the solvers required for qualitative and quantitative[1] backward/forward reasoning. Solvers have been compiled and tested under Windows Xp, Linux, and MacOS X. Quantitative backward reasoning uses Lingo as solver. Lingo is a commercial optimization modeling tool commercial produced and distributed by Lindo System. In the GR-Tool we use Lingo version 8.0.

7. Conclusions

In this paper, we have presented a formal framework for modeling and reasoning with goal models. The framework presents a formalization of *i**/Tropos goal models and a set of axioms that describe the possible relations among goals. On the base of such

[1] This paper does not present quantitative reasoning mechanisms for goal models; these have been extensively discussed in [8, 9, 10].

formalization, we have introduced two different forms of qualitative reasoning on goal models: forward and backward reasoning. Given a goal model and an initial assignment of evidence for the satisfaction of some goals, forward reasoning focuses on how to propagate this evidence (labels) forward, towards root goals. Backward reasoning, on the other hand, focuses on finding a label assignment for leaf nodes of a goal graph that satisfies/denies all root goals. We have also presented an example of how to use our framework and the tool we have developed to support the whole approach. As future work, we are interested to work in two directions. Firstly, we would like to integrate better goal analysis in the Tropos methodological process, identifying with real case studies when and how the two forms of reasoning can be applied during the development of a software system. Secondly, we are interested on investigate the possibility of extend the framework with different forms of analysis, such as for example risk analysis.

Acknowledgements

This work has been partly supported by the projects EU-SERENITY, FIRB-ASTRO, PAT-MOSTRO, and PAT-STAMPS.

Endnotes

1 A Boolean formula is in CNF if and only if it is in the form $\wedge_i \vee_j l_{ij}$ where l_{ij} are literals. A disjunction $\vee_{ji} l_{ji}$ is called a *clause*. A one-literal clause is called *unit clause*.

2 Broadly speaking, Δ_2^p is the class of problems requiring a polynomial amounts of calls to a procedure solving an NP problem.

3 Recall that by the very definition of goal graphs, every loop contains at least one leaf goal.

4 This is a revised version of a case study originally presented in [2].

References

1. P. Bresciani, P. Giorgini, F. Giunchiglia, J. Mylopoulos, and A. Perini. Tropos: An agent oriented software development methodology. Journal of Autonomous Agents and Multi-Agent Systems, 8(3):203–236, May 2004.
2. Jaelson Castro, Manuel Kolp, and John Mylopoulos. Towards Requirements-Driven Information Systems Engineering: The Tropos Project. Information Systems, 27(6):365–389, 2002.
3. Lawrence K. Chung, *Representing and Using Non-Functional Requirements: A Process-Oriented Approach*, PhD thesis, Department of Computer Science, University of Toronto, 1993.
4. S. A. Cook. The complexity of theorem proving procedures. In 3rd Annual ACM Symposium on the Theory of Computation, pages 151-158, 1971.
5. J. Conallen. *Building Web Applications with UML*. Addison-Wesley, 2000.
6. A. Dardenne, A. van Lamsweerde and S. Fickas, "Goal-Directed Requirements Acquisition", *Science of Computer Programming*, Vol. 20, 1993, 3-50.
7. M. Davis, G. Logemann, and D. Loveland. A machine program for theorem proving. Journal of the ACM, 5(7), 1962.
8. P. Giorgini, J. Mylopoulos, and R. Sebastiani. Goal-Oriented Requirements Analysis and Reasoning in the Tropos Methodology. *Engineering Applications of Artificial Intelligence*, Elsevier, Volume 18/2, marzo 2005.

9. P. Giorgini, E. Nicchiarelli, J. Mylopoulos, and R.Sebastiani. Reasoning with Goal Models. In Proc. Int. Conferente of Conceptual Modeling (ER'02), LNCS, Tampere, Finland, October 2002. Springer.
10. P. Giorgini, E. Nicchiarelli, J. Mylopoulos, and R.Sebastiani. Formal Reasoning Techniques for Goal Models. *Journal of Data Semantics*, 1, October 2003. Springer.
11. A. van Lamsweerde, "Requirements Engineering in the Year 00: A Research Perspective", Keynote paper, *Proc. ICSE'2000 - 22nd Intl. Conf. on Software Engineering*, 2000.
12. P. Liberatore. Algorithms and Experiments on Finding Minimal Models. Technical report, DIS, University of Rome "La Sapienza", December 2000. Available at http://www.dis.uniroma1.it/~liberato/mindp/.
13. M. W. Moskewicz, C. F. Madigan, Y. Z., L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In Design Automation Conference, 2001.
14. Mylopoulos, J., Chung, L. and Nixon, B., "Representing and Using Non-Functional Requirements: A Process-Oriented Approach," *IEEE Transactions on Software Engineering 18(6)*, June 1992, 483-497.
15. N. Nilsson. Problem Solving Methods in Artificial Intelligence. McGraw Hill, 1971.
16. C. Potts and Bruns, G., "Recording the Reasons for Design Decisions", Proceedings International Conference on Software Engineering, Singapore, 1988.
17. C. Rolland, "Reasoning with Goals to Engineer Requirements", Proceedings Third International Conference on Enterprise Information Systems, Setubal Portugal, July 2001.
18. R. Sebastiani, P. Giorgini, and J. Mylopoulos. Simple and Minimum-Cost Satisfiability for Goal Models. In *Proc. Int. conference on Advanced Information Systems Engineering, CAISE'04*, volume 3084 of *LNCS*, pages 20–33. Springer, June 2004.

19. E. Yu. *Modelling Strategic Relationships for Process Reengineering*. PhD thesis, University of Toronto, Department of Computer Science, 1995.

20. Lintao Zhang and Sharad Malik. The Quest for Efficient Boolean Satisfiability Solvers. Proceedings CAV'02, LNCS no 2404, Springer, 2002, 17-36.