klase

WP2.1 - Approccio trasformazionale incrementale per l'analisi dei requisiti

Università di Trento
ITC-irst

Abstract.
This deliverable presents the results of the WP2-Task 2.1 and notably definition of the incremental transformational approach for the functional and non-functional requirements analysis in order to move from a conceptual model to another, both inside the same phase (intra-phase) and between two subsequent phases (inter-phase). In particular, it defines a set of transformation primitives that, starting from the actors and their dependencies, allow to explore all the possible alternative ways that the actors can adopt in order to satisfy the dependencies in which they are involved.

Document Identifier	Deliverable Dm.n
Project	MIUR-FIRB project RBNE0195K5 "Knowledge Level Automated Software Engineering"
Version	v1.0
Date	November 5, 2006
State	Final
Distribution	Public

Acknowledgements.

This document is part of a research project funded by the FIRB 2001 Programme of the “Ministero dell’Istruzione, dell’Università e della Ricerca” as project number RBNE0195K5.

The partners in this project are: Istituto Trentino di Cultura (Coordinator), Università degli Studi di Trento, Università degli Studi di Genova, Università degli Studi di Roma “La Sapienza”, DeltaDator S.p.A..

Executive Summary

This deliverable presents the results of the WP2-Task 2.1 and notably definition of the incremental transformational approach for the functional and non-functional requirements analysis in order to move from a conceptual model to another, both inside the same phase (intra-phase) and between two subsequent phases (inter-phase). It defines a set of transformation primitives that, starting from the actors and their dependencies, allow to explore all the possible alternative ways that the actors can adopt in order to satisfy the dependencies in which they are involved.

In particular, we present the redefinition of the transformation system for early requirements analysis, proposed in [BPG+02], in terms of a Graph Transformation System. Then, we focus on the visual modeling in *Tropos* and a set of specific issues related to the development of a CASE tool supporting it. Finally, we investigate the transformation of Security Requirements through Planning.

Contents

Chapter 1

Introduction

Tropos [PBG+01, gio01, ea06] is an agent-oriented software development methodology in which AI derived mentalistic notions, such as *actors*, *goals*, *softgoals*, *plans*, *resources*, and *intentional dependencies* are used in all the phases of software development, from the first phases of early analysis down to the actual implementation. A crucial role is given to the earlier analysis of requirements that precedes prescriptive requirements specification. In particular, aside from the understanding of *how* the intended system will fit into the organizational setting, and *what* the system requirements are, Tropos addresses also the analysis of the *why* the system requirements are as they are, by performing an in-depth justification with respect to the organizational goals.

Tropos rests on five main phases for agent based systems development: *early requirements analysis*, *late requirements analysis*, *architectural design*, *detailed design*, and *implementation*. An incremental and iterative development process is adopted inside each phase and among different phases, in particular during early requirements analysis and late requirements analysis [BS02].

Early requirements analysis is concerned with the understanding of the problem by studying an existing organizational setting: the requirement engineer models and analyzes the desires and the intentions of the stakeholders, and states their intentional dependencies. Desires, intentions, and dependencies are modeled as goals and as softgoals which, through a goal-oriented analysis, provide the rationale for the specification of the functional and non-functional requirements of the system-to-be. The output of this phase is an organizational model which includes relevant actors and their respective dependencies. Actors in the organizational setting are characterized by having goals that each single actor, in isolation, would be unable —or not as competent— to achieve. The goals are achievable in virtue of reciprocal means-end knowledge and dependencies.

In an earlier work [BPG+02], it has been proposed a characterization of the process of early requirements analysis, defining it in terms of applying transformations to the model of the system. In particular, the work focused on the definition of the transformations that can be applied for refining an initial Tropos model to a final one, working incremen-

tally. This is a very basic issue in defining a new methodology, as for instance proposed in [BCN92] for Entity-Relationship schema design, in [DvLF93] for goal oriented requirements analysis, and in [MCL+01] for functional and non-functional requirements analysis. Therefore, one of the focuses of this deliverable is on the redefinition of the transformation system for early requirements analysis, proposed in [BPG+02], in terms of a Graph Transformation System.

From another point of view, visual modeling is a common practice in software development, especially in Object Oriented software development where a standard modeling language (the Unified Modeling Language - UML [gro]) has been proposed and several CASE tools to support of the modeling activity are available [CAS]. This led also to the adoption of visual modeling as a core discipline in industrial software development processes, such as the Rational Unified Process [RUP]. Nevertheless, some interesting research problems are still open, such as how to provide a common semantics for structural and behavioral UML diagrams [TE00, GZK03], how to enhance OCL constraints for UML class diagrams with Graph Rewriting (GR) concepts [Sch01]. GR plays an important role in solving some of them, as discussed in previous GT-VMT workshops [GT-]. Therefore, this deliverable studies the visual modeling in *Tropos* and a set of specific issues related to the development of a CASE tool supporting it.

The design of secure and trusted software that meets stakeholder needs is an increasingly hot issue in Software Engineering (SE). This quest has led to refined Requirements Engineering (RE) and SE methodologies so that security concerns can be addressed during the early stages of software development (e.g., Secure Tropos vs i*/Tropos, UMLsec vs UML). Moreover, industrial software production processes have been tightened to reduce the number of existing bugs in operational software systems through code walkthroughs, security reviews, etc. Further, the complexity of present software is such that all methodologies come with tools for automation support.

The tricky question in such a setting is what kind of automation is needed? Almost fifty years ago the idea of actually deriving code directly from the specification (such as that advocated in [MW80]) started a large programme for deductive program synthesis[1], that is still active now [Cal97, Ell06, MT, RB05]. However, proposed solutions are largely domain-specific, require considerable expertise on the part of their users, and in some cases do not actually guarantee that the synthesized program will meet all requirements stated up front [Ell06].

In most SE methodologies the designer has tools to report and verify the final choices (be it goal models in KAOS, UML classes, or Java code), but not actually the possibility of automatically exploring design alternatives (i.e., the *potential choices* that the designer may adopt for the fulfillment of system actors' objectives) and finding a satisfactory one. Conceptually, this automatic selection of alternatives is done in deductive program syn-

[1] A system goal together with a set of axioms are specified in a formal specification language. Then the system goal is proved from the axioms using a theorem prover. A program for achieving the goal is extracted from the proof of the theorem.

thesis: theorem provers select appropriate axioms to establish the system goal. Instead, we claim that the automatic selection of alternatives should and indeed can be done during the very early stages of software development. After all, the automatic generation of alternatives is most beneficial and effective during these stages. There are good reasons for this claim. Firstly, during early stages the design space is large, and a good choice can have significant impact on the whole development project. Supporting the selection of alternatives could lead to a more thorough analysis of better quality designs with respect to security and trust. Secondly, requirements models are by construction simpler and more abstract than implementation models (i.e., code). Therefore, techniques for automated reasoning about alternatives at the early stages of the development process may succeed where automated software synthesis failed.

Since the overall goal is to design a secure system we have finally singled out the Secure Tropos methodology [GMMZ05a] as the last target for our work in this deliverable. Its primitive concepts include those of Tropos and i* [CKM02], but also concepts that address security concerns, such as ownership, permission and trust.

1.1 Technical contribution

In this deliverable we redefine the *Tropos* analysis process in terms of a Graph Transformation System and we provide an algorithm for driving the process of *Tropos* diagram generation. An example execution of the algorithm is also presented.

We describe our approach, based on Graph Rewriting (GR), to support the visual modeling process in *Tropos*. We give examples of the rewriting rules which specify the syntax of *Tropos* and discuss how this graph rewriting rule-set can support an analyst in building correct models (that is models consistent with the *Tropos* language). Moreover we consider how GR permits to adopt modeling language variants in a flexible way and to support an incremental modeling process.

Automated reasoning mechanisms for requirements and software verification are by now a well-accepted part of the design process, and model driven architectures support the automation of the refinement process. We claim that we can further push the envelope towards the automatic exploration and selection among design alternatives and show that this is concretely possible for Secure Tropos, a requirements engineering methodology that addresses security and trust concerns. In Secure Tropos, a design consists of a network of actors (agents, positions or roles) with delegation/permission dependencies among them. Accordingly, the generation of design alternatives can be accomplished by a planner which is given as input a set of actors and goals and generates alternative multi-agent plans to fulfill all given goals. We validate our claim with a case study using a state-of-the-art planner.

1.2 Plan of the deliverable

The rest of the deliverable is organized as follows. In the second chapter the main focus is on the redefinition of the transformation system for early requirements analysis, proposed in [BPG+02], in terms of a Graph Transformation System. This provides the necessary machinery to perform precise inspections of the process of early requirements analysis, and allows us to distinguish among different strategies for the execution of the process.

The third chapter focuses on the visual modeling in *Tropos* and a set of specific issues related to the development of a CASE tool at support of visual modeling in *Tropos*, such as, how to assure that a model is correct with respect to a given syntax, how to support the building of consistent instances of a given model, and how to provide services for automatic restructuring of the model like the application of different patterns and refactoring.

Finally, the main focus of the last chapter is on the transformation of Security Requirements through Planning.

Chapter 2

The Tropos Analysis Process as Graph Transformation System

2.1 The Transformational Based Approach

Tropos rests on the use of a conceptual model of the system-to-be —and of the organizational environment in which the system will operate— specified by a modeling language based on Eric Yu's ***i**** paradigm [Yu95]. This paradigm offers actors, goals, and actor dependencies as primitive concepts for modeling an application during the requirements analysis. During early requirements analysis, the requirements engineer models and analyzes the intentions of the stakeholders. Following ***i****, in Tropos the stakeholders' intentions are modeled as goals which, through some form of goal-oriented analysis, eventually lead to the functional and non-functional requirements of the system-to-be. The detailed analysis of the system requirements is then dealt with during the late requirement analysis phase.

Early requirements are assumed to involve social actors who depend on each other for goals to be achieved, tasks to be performed, and resources to be provided. Tropos includes *actor diagrams* for describing the network of social dependency relationships among actors, as well as *rationale diagrams* for analyzing and trying to fulfill goals through a means-ends analysis.

These primitives are formalized using intentional concepts from AI, such as goal, belief, ability, and commitment. Actor and rationale diagrams may be combined in order to convey a global view on the model they describe together. An example of actor and rationale diagrams is showed in Figure 2.1. An organization analysis is depicted, in which two relevant actors —the `Citizen` and the `PAT` (Autonomous Province of Trento)— depend upon each other in order to fulfill the citizen's goal of having a system for accessing cultural information (`system available`) and the citizen's softgoal[1] of having taxes

[1] Softgoals are mainly used for specifying additional qualities or vague requirements.

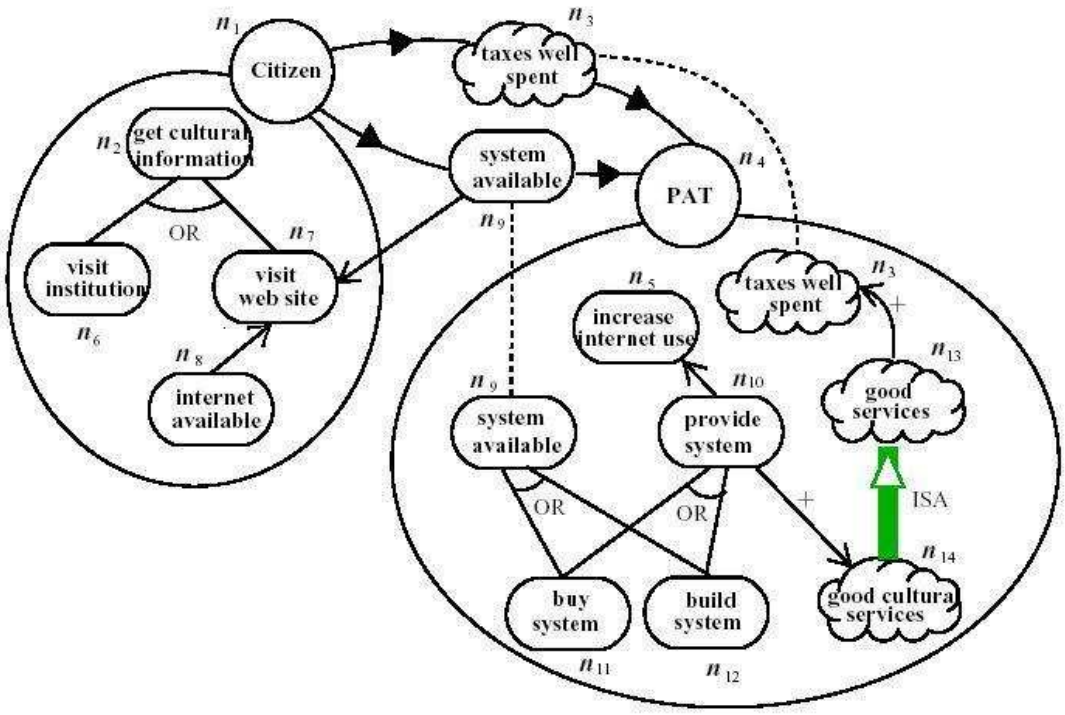


Figure 2.1: A Tropos actor diagram with rationale analyses for two actors (`Citizen` and `PAT`).

well spent (`taxes well spent`). One possible analysis of this situation produced the diagram of Figure 2.1, as illustrated in [BPG+02], where `Citizen` is associated with the initial relevant goal `get cultural information`, that is then OR-decomposed into the two subgoals `visit institution` and `visit web site`. One means for fulfilling the goal `visit web site` is to have a web cultural information system available (goal `system available`). This last goal is then similarly analyzed from the point of view of the actor `PAT`, as well as other goals and softgoals. For example, the positive or negative contribution of softgoals (only positive in the example) can be introduced, as well as ISA hierarchies. A more detailed description of this and related examples can be found in[PBG+01, gio01, BPG+02]

Early requirement analysis and late requirement analysis[2] are iterative processes, at each step of which details are incrementally added and rearranged, introducing initially only few actors, goals, softgoals, and dependencies, and adding then more and more elements. The details added at each step are aimed at representing increasing knowledge and insight on the problem and its environment, and their introduction corresponds to a

[2]For most of the aspects analyzed in this chapter, early requirement analysis and late requirement analysis can be considered as being carried on in the same way. Thus, in the next pages we will consider the early requirement analysis process, only.

deeper and more precise understanding of the requirements. It is relevant to observe that, with regard to the transition from early requirements to late requirements, this approach is particularly fruitful. In fact, making the system-to-be requirements correspond to the real state of affairs of the organizational setting is much more natural, because the organizational goals are directly linked to the system requirements, so that the later are justified by the former. It is in this way that not only the *how* and the *what* dealt also by standard Requirement Engineering techniques is described, but also the *why*.

In this context, for both early and late requirements analyses [BS02], the process of conceptual modeling of the environmental setting can be described in terms of simple transformations of subsequent versions of the model. Each of these transformations allows for the progressive introduction of more structure and details into the model. In other words, by iterating the application of transformations, the engineer can progressively move toward the final complete model, going through subsequent, more and more precise and detailed, versions.

In [BPG$^+$02], the most relevant transformations have been introduced, and in particular:

1. **Goal transformations**, which allow us to perform goal analysis by introducing relationships between goals, or actors and goals. We distinguish:
   - *Goal decomposition transformations*, which allow for the decomposition of a goal into and/or subgoals, such that the achievement of one subgoal (for the *OR-decomposition*) or all subgoals (for the *AND-decomposition*) implies the root goal achievement.
   - *Precondition goal transformations*, which allow us to list a set of necessary (but not sufficient) preconditions in terms of other goals. Precondition goals *enable* the achievement of the higher level goal. The resulting analysis has to be somehow completed with more elements derived from further goal or task analyses, possibly during later Tropos phases, like the late requirement analysis.
   - *Goal delegation transformations*, which are aimed at allowing the model to express the change of responsibility in goal fulfillment. The goal delegation transformation can be applied to a goal and to the actor it is assigned to.
   - *Goal generalization transformations*, which allow us to introduce an ISA hierarchy among two goals. The same hierarchy must also hold between the two actors the two goals are assigned to, and, when it is the case, the four actors that participate at the two (goal) dependency relationships.
2. **Softgoal transformations**, which allow us to perform softgoal analysis. They are similar to the goal transformations (with the exclusion of the precondition transformation). To deal with the so called *contribution analysis* (see [MCL$^+$01, PBG$^+$01]), one additional transformation is used:

- *Contribution transformations*, which allow us to specify whether a goal or soft-goal contributes to some other softgoal or, starting from the other side, whether there is some goal or soft-goal that contributes positively or negatively to the satisfaction of the softgoal.

3. **Actor Transformations:** also some Actor Transformations have been described in [BPG+02], but they are not dealt with in this chapter.

Transformations can be considered as the building blocks of the engineer's activity, and the way they are used can be analyzed also with respect to the strategic role they play in the design process.

Specific (local) strategies can be defined as *top-down* or as *bottom-up*. Applying transformations in a top-down way allows the engineer to analyze high level conceptual elements (actors, goals, softgoals) by adding details in terms of relationships (specialization, decomposition, softgoal contribution, etc.) or dependencies with respect to other conceptual elements. Vice versa, bottom-up transformation applications allow us to aggregate finer grain conceptual elements in order to express their contribution —compositional, hierarchical, functional or non-functional— to other, somehow more generic, conceptual elements.

A clear definition of transformation applications and of a way to analyze applications sequences is important in order to allow us to compare different strategies for the design process. Of course, engineering activity cannot be totally reduced in formal steps, considering that it includes a considerable informal and human contribution. But separating diagram transformations from other decisional elements is certainly a first step to allow for a better description and analysis of the process.

In the following sections we present a first attempt to formalize the diagram transformations listed above and in [BPG+02] in terms of a Graph Transformation Systems. Also, we provide some first observations than can be derived from the comparison of the execution of two different algorithms for the process of design development.

Toward this aim, we will refer to the task of developing the final diagram shown in Figure 2.1, that is derived from a case study presented more in detail in [BPG+02].

2.2 The analysis process as a graph transformation system

In the previous section an intuition of the Tropos early and late requirement analysis process has been given.

The real focus of this chapter is in providing a more formal description of the process. In this section, this will done by adopting notions of Graph Transformation Systems as presented in [AEH+99].

Node type restrictions for different types of edges		
$l_1(e)$	$l_1(s(e))$	$l_1(t(e))$
$outgoing\text{-}dependency$	$actor$	$goal, softgoal, plan, resource$
$incomming\text{-}dependency$	$goal, softgoal, plan, resource$	$actor$
$and\text{-}decomposition$	$l_2(t(e))$	$goal, softgoal, plan$
$or\text{-}decomposition$	$l_2(t(e))$	$goal, softgoal, plan$
$precondition$	$goal$	$goal$
$generalization$	$l_2(t(e))$	$goal, softgoal, task, actor$
$assigment$	$goal, softgoal, resource, plan$	$actor$
$positive\text{-}contribution$	$softgoal, goal$	$softgoal$
$negative\text{-}contribution$	$softgoal, goal$	$softgoal$
$aggregation$	$actor$	$actor$

Table 2.1: Restrictions for the edges in a valid Tropos diagram.

A Tropos actor or rationale diagram can be simply seen as a special case of *labeled directed graph*, namely a 5-tuple $G = \langle \mathcal{N}, \mathcal{E}, s, t, l \rangle$, where $\mathcal{N}$ is a finite set of nodes, each pair of which can be connected by one or more edges of the finite set $\mathcal{E}$; s and t are two functions:

$$s, t : \mathcal{E} \longrightarrow \mathcal{N}$$

that assign to each edge the source and the target node, respectively; l is a labeling function for each node and edge. For Tropos actor diagrams and rationale diagrams, it can be assumed that:

$$l : \mathcal{N} \cup \mathcal{E} \longrightarrow \mathtt{T} \times \mathtt{L}$$

where T is the set of possible types of nodes and edges, T={*actor*, *goal*, *softgoal*, *resource*, *plan*, *and-decomposition*, *or-decomposition*, *precondition*, *generalization*, *assignment*, *outgoing-dependency*, *incoming-dependency*, *positive-contribution*, *negative-contribution*, *aggregation*}[3], and L is any set of desirable identifiers (e.g., generic ASCII strings). Moreover, it is assumed that for each Tropos actor or rationale diagram $G = \langle \mathcal{N}, \mathcal{E}, s, t, l \rangle$[4], $l_2(\mathcal{E}) = \{\epsilon\}$[5], $l_1(\mathcal{N}) \subseteq$*{actor, goal, softgoal, resource, plan}* and $l_1(\mathcal{E}) \subseteq$ *and-decomposition, or-decomposition, outgoing-dependency, incoming-dependency, precondition, generalization, assignment, positive-contribution, negative-contribution, aggregation*}.[6] Furthermore, for G to be a *valid* Tropos diagram, the restrictions listed in Table 2.1, on the types of the nodes connected by an edge, must be observed.

[3]The assignment edge is not explicitly visualized in Tropos diagrams. Instead, this notion is reproduced either by "attaching" the goal/softgoal/plan/resouce to the actor, or by placing these nodes inside a "balloon" that represent the actor's context.

[4]In the following, when no ambiguity arise, $\mathcal{N}, \mathcal{E}, s, t, l$ will always be used as the components of G.

[5]l_2 selects the second component of l, namely, the identifier; similarly, l_1 select the first component, namely the type. ϵ is the empty string.

[6]Self-understandable abbreviations will be used when needed.

2.2.1 Graph transformation system

The notion of a *graph transformation rule* allows us to give a formal account of different kinds of computation applied to graphs. A simple, yet complete, definition can be found in [AEH+99]. In our case, the following, less general, notion of graph transformation rule is sufficient.

A *graph transformation rule* is a pair $r = \langle L, R \rangle$, where L and R are graphs with a well defined and non-empty intersection[7], also called left-hand-side (LHS) and right-hand-side (RHS) of the rule. The application of rule r to a graph G yields a new graph H obtained as follow.

1. Chose an isomorphism[8] i (called *occurrence isomorphism*) from L onto a subgraph G' of G; (a graph G' is a subgraph of a graph G iff $G' \cap G$ is well defined and $G' \cap G = G'$).

2. Delete from G the images of L with no counter-images in $L \cap R$, and obtain the *context graph* $D = G \setminus i(L \setminus R)$.

3. Add to D the images of the items of R not already in D; this yields $H = D \cup i(R \setminus L)$.

From this definition it follows that the possible rule application is not always unique for a given rule and a given graph, because different occurrence isomorphisms may apply. If a graph H is obtained from graph G by the application of rule r, we will write:

$$G \stackrel{r}{\Rightarrow} H.$$

A *Graph Transformation System* is a set $P = \{r_1, \ldots, r_n\}$ of graph transformation rules. A graph H is said to be derivable from graph G by means of a sequence of applications of rules in P if:

$$G \stackrel{r_1}{\Rightarrow} G_1 \stackrel{r_2}{\Rightarrow} G_2 \ldots \stackrel{r_{m-1}}{\Rightarrow} G_{m-1} \stackrel{r_m}{\Rightarrow} H$$

with $r_i \in P$, for $1 \leq i \leq m$.

We will also write $G \stackrel{P}{\Longrightarrow} H$, or $\stackrel{P}{\Longrightarrow} H$ in the case G is the empty graph.

Notice that, given a graph transformation system P and an initial graph G, the derivation process is non deterministic, due to both the occurrence choice (as already noted

[7] A graph intersection $G = G' \cap G''$, where $G' = \langle \mathcal{N}', \mathcal{E}', s', t', l' \rangle$ and $G'' = \langle \mathcal{N}'', \mathcal{E}'', s'', t'', l'' \rangle$, is the graph $G = \langle \mathcal{N}, \mathcal{E}, s, t, l \rangle$, such that $\mathcal{N} = \mathcal{N}' \cap \mathcal{N}''$, $\mathcal{E} = \mathcal{E}' \cap \mathcal{E}''$, and s, t, and l, are either s', t', and l', or s'', t', and l'' restricted to the domains of G. Of course, it is required that $s' = s''$, $t' = t''$, and $l' = l''$ when applied to any element of $\mathcal{N}$ or $\mathcal{E}$, in order that the intersection is well defined.

[8] More in general, homomorphisms preserving s, t, and l could be considered, but for sake of simplicity only isomorphisms will be considered in the examples of the present chapter.

above) and the rule choice, at each step. The length of the derivation is another open parameter. Thus, different graphs may be derived from G by P, as well as, supposed that $G \stackrel{P}{\Longrightarrow} H$, there may be different possible derivations of H.

The Tropos actor and rationale diagram definition process may be described in term of a graph transformation system. The Tropos graph transformation system is described by the set of rules reported in Table 2.2.

2.2.2 The Tropos diagram generation algorithm

```
BEGIN
  'initialize graph' G (* in general empty *)
  REPEAT
    'chose an applicable rule' r=<L,R> IN P;
    'chose an occurrence isomorphism' i 'for the application of' r;
    G := (G \ i(L\R)) UNION i(R\L)
  UNTIL G = 'desired graph'  OR  'no applicable rules left';
  IF G = 'desired graph' THEN RETURN(G)
                         ELSE FAIL
END
```

Figure 2.2: The algorithm for the Tropos graph generation.

Using the graph transformation system given in Table 2.1, the generic algorithm for driving the process of Tropos graph (or Tropos diagram) generation is given in Figure 2.2.

The `'chose an applicable rule'` and `'chose an occurrence'` steps correspond to the non-deterministic choices that, as already mentioned, are intrinsic in the definition of rule-system application. The test `'desired graph'` is meant to verify when a reasonably detailed diagram is obtained. This can be done by combining informal decision criteria on the satisfiability of single goal, softgoal, or plan leaf nodes, together with label propagation algorithm that allows us to compute the satisfaction of (non-leaf) nodes, starting from the satisfaction of leaf nodes, as, e.g., described in [MCN92].

Several constraints and heuristics may be introduced in order to *control* this kind of non determinism. Among them we can list:

- assign priority to rules
- assign a precedence ordering among different rules (that may depend on the context of execution)
- define an absolute ordering among categories of rules.

As a first attempt, we consider the possibility of distinguishing among different categories of rules, and in particular, we group the rules of Table 2.2 in three categories. The

Graph Transformation System for Tropos diagrams	
name: A-I (Actor Introduction)	category: *introduction*
LHS: $\langle\{\},\{\},\{\},\{\},\{\}\rangle$	
RHS: $\langle\{n_1\},\{\},\{\},\{\},\{n_1 \mapsto \langle ACT, *\rangle\}$	
name: G-I (Goal Introduction)	category: *introduction*
LHS: $\langle\{n_1\},\{\},\{\},\{\},\{n_1 \mapsto \langle ACt, *\rangle\}\rangle$	
RHS: $\langle\{n_1,n_2\},\{e_1\},\{e_1 \mapsto n_2\},\{e_1 \mapsto n_1\},\{n_1 \mapsto \langle ACT,*\rangle, n_2 \mapsto \langle G,*\rangle\, e_1 \mapsto \langle ASS,\epsilon\rangle\}$	
name: SG-I (SoftGoal Introduction)	category: *introduction*
LHS: $\langle\{n_1\},\{\},\{\},\{\},\{n_1 \mapsto \langle ACT, *\rangle\}\rangle$	
RHS: $\langle\{n_1,n_2\},\{e_1\},\{e_1 \mapsto n_2\},\{e_1 \mapsto n_1\},\{n_1 \mapsto \langle ACT,*\rangle, n_2 \mapsto \langle SG,*\rangle\, e_1 \mapsto \langle ASS,\epsilon\rangle\}$	
name: G-DEC-& (Goal AND Decomposition)	category: *analysis*
LHS: $\langle\{n_0,n_1,\ldots,n_n\},\{\},\{\},\{\},\{n_i \mapsto \langle G,*\rangle\}\rangle$	
RHS: $\langle\{n_0,n_1,\ldots,n_n\},\{e_1,\ldots e_n\},\{e_i \mapsto n_i\},\{e_i \mapsto n_0\},\{n_i \mapsto \langle G,*\rangle, e_i \mapsto \langle \&\text{-}DEC,\epsilon\rangle\}$	
name: G-DEC-OR (Goal OR Decomposition)	category: *analysis*
LHS: $\langle\{n_0,n_1,\ldots,n_n\},\{\},\{\},\{\},\{n_i \mapsto \langle G,*\rangle\}\rangle$	
RHS: $\langle\{n_0,n_1,\ldots,n_n\},\{e_1,\ldots e_n\},\{e_i \mapsto n_i\},\{e_i \mapsto n_0\},\{n_i \mapsto \langle G,*\rangle, e_i \mapsto \langle OR\text{-}DEC,\epsilon\rangle\}$	
name: G-P (Goal Precondition)	category: *analysis*
LHS: $\langle\{n_1,n_2\},\{\},\{\},\{\},\{n_i \mapsto \langle G,*\rangle\}\rangle$	
RHS: $\langle\{n_1,n_2\},\{e_1\},\{e_1 \mapsto n_2\},\{e_1 \mapsto n_1\},\{n_i \mapsto \langle G,*\rangle, e_1 \mapsto \langle Precond,\epsilon\rangle\}$	
name: G-G (Goal Generalization)	category: *analysis*
LHS: $\langle\{n_1,n_2\},\{\},\{\},\{\},\{n_i \mapsto \langle G,*\rangle\}\rangle$	
RHS: $\langle\{n_1,n_2\},\{e_1\},\{e_1 \mapsto n_2\},\{e_1 \mapsto n_1\},\{n_i \mapsto \langle G,*\rangle, e_1 \mapsto \langle ISA,\epsilon\rangle\}$	
name: SG-C-$SG(+)$ (SoftGoal-SoftGoal positive Contribution)	category: *analysis*
LHS: $\langle\{n_1,n_2\},\{\},\{\},\{\},\{n_i \mapsto \langle SG,*\rangle\}\rangle$	
RHS: $\langle\{n_1,n_2\},\{e_1\},\{e_1 \mapsto n_2\},\{e_1 \mapsto n_1\},\{n_i \mapsto \langle SG,*\rangle, e_1 \mapsto \langle PosContr,\epsilon\rangle\}$	
name: SG-G (SoftGoal Generalization)	category: *analysis*
LHS: $\langle\{n_1,n_2\},\{\},\{\},\{\},\{n_i \mapsto \langle SG,*\rangle\}\rangle$	
RHS: $\langle\{n_1,n_2\},\{e_1\},\{e_1 \mapsto n_2\},\{e_1 \mapsto n_1\},\{n_i \mapsto \langle SG,*\rangle, e_1 \mapsto \langle ISA,\epsilon\rangle\}$	
name: G-DEL (Goal Delegation)	category: *delegation*
LHS: $\langle\{n_1,n_2,n_3\},\{e_1\},\{e_1 \mapsto n_2\},\{e_1 \mapsto n_1\},\{n_{1,3} \mapsto \langle Act,*\rangle, n_2 \mapsto \langle G,*\rangle, e_1 \mapsto \langle Ass,\epsilon\rangle\}\rangle$	
RHS: $\langle\{n_1,n_2,n_3\},\{e_2,e_3,e_4\},\{e_2 \mapsto n_1, e_{3,4} \mapsto n_2\},\{e_2 \mapsto n_2, e_{3,4} \mapsto n_3\},$ $\{n_{1,3} \mapsto \langle Actor,*\rangle, n_2 \mapsto \langle Goal,*\rangle, e_2 \mapsto \langle OutDep,\epsilon\rangle, e_3 \mapsto \langle InDep,\epsilon\rangle, e_4 \mapsto \langle Ass,\epsilon\rangle\}$	
name: SG-DEL (SoftGoal Delegation)	category: *delegation*
LHS: $\langle\{n_1,n_2,n_3\},\{e_1\},\{e_1 \mapsto n_2\},\{e_1 \mapsto n_1\},\{n_{1,3} \mapsto \langle Act,*\rangle, n_2 \mapsto \langle SG,*\rangle, e_1 \mapsto \langle Ass,\epsilon\rangle\}\rangle$	
RHS: $\langle\{n_1,n_2,n_3\},\{e_2,e_3,e_4\},\{e_2 \mapsto n_1, e_{3,4} \mapsto n_2\},\{e_2 \mapsto n_2, e_{3,4} \mapsto n_3\},$ $\{n_{1,3} \mapsto \langle Actor,*\rangle, n_2 \mapsto \langle SoftGoal,*\rangle, e_2 \mapsto \langle OutDep,\epsilon\rangle, e_3 \mapsto \langle InDep,\epsilon\rangle, e_4 \mapsto \langle Ass,\epsilon\rangle\}$	

Table 2.2: Transformation rules: the maps of functions s, t, and l are fully listed. Abbreviations for some type names are used. The wild-char $*$ stands for any string.

first, including A-I, G-I, and SG-I, is characterized by the introduction of a new node in the RHS graph. The second, including G-DEC-$\&$, G-DEC-OR, SG-C-$SG(+)$, and so on, is characterized by the introduction of new edge(s). The last category includes the rules G-DEL and SG-DEL, that imply the deletion of an assignment edge in the LHS graph, and its replacement in the RHS graph with three new edges: two used to form a chain of the type $actor \rightarrow dependum \rightarrow actor$[9] (where $dependum$ is either $goal$ or $softgoal$) and a third to state a new assignment for the dependum to the second actor, thus completing the definition of a kind of dependum delegation. We call these three categories *introduction rules*, *analysis rules*, and *delegation rules*, respectively.

The new version of the algorithm is given in Figure 2.3. The general idea is to apply first all the applicable rules of a category, and then proceed with rules of the next category. Indeed, it soon turns out that it may be convenient to allow for simple exceptions. Let's consider the case of analysis rules: in many cases their application (especially for decomposition rules) require the existence of some nodes (e.g., the subgoals or the sub-softgoals) that may be not already be present in diagram. In order to avoid delaying the application of the analysis rule until the next REPEAT loop, it may be preferable to allow for the interleaving of the appropriate introduction rule. In particular, this may be considered a quite standard case when the analysis rule is applied in a Top-Down direction.

The outer REPEAT loop is necessary because the application of rules of one category may make rules of other categories applicable as well, although they were not so before.

Below, it is shown the use of the algorithm to produce the analysis of Figure 2.1.

Outer REPEAT:

First LOOP:

INTRODUCTION RULES:[10]

$\langle\{\},\{\}\rangle \overset{A\text{-}I}{\Rightarrow}_1 \langle\{n_1\},\{\}\rangle$
$\overset{G\text{-}I}{\Rightarrow}_2 \langle\{n_1,n_2\},\{n_2 \rightarrow n_1\}\rangle$
$\overset{SG\text{-}I}{\Rightarrow}_3 \langle\{n_1,n_2,n_3\},\{n_2 \rightarrow n_1, n_3 \rightarrow n_1\}\rangle$
$\overset{A\text{-}I}{\Rightarrow} \langle\{n_1,n_2,n_3,n_4\},\{n_2 \rightarrow n_1, n_3 \rightarrow n_1\}\rangle$
$\overset{G\text{-}I}{\Rightarrow} \langle\{n_1,n_2,n_3,n_4,n_5\},\{n_2 \rightarrow n_1, n_3 \rightarrow n_1, n_5 \rightarrow n_4\}\rangle$

ANALYSIS RULES:[11]

[9] We use an arrow between two nodes ($n_1 \rightarrow n_2$) to shortly indicate the presence of an edge between them.

[10] In the following derivations, for each generated graph only $\mathcal{N}$ is explicitly listed. The elements of $\mathcal{E}$ are not named; instead the two maps s and t are given in compact form by writing, e.g., $n_1 \rightarrow n_2$: in this case we mean that $s(e) = n_1$ and $t(e) = n_2$. The Function l can be easily read on Figure 2.1.

[11] To make our notation more compact, at each transformation step we will denote the set of nodes so far included in the graph simply with $\mathcal{N}$, and the set of edge so far included in the graph simply with

```
BEGIN
  'initialize' graph G (* in general empty *)
  REPEAT

    WHILE G <> 'desired graph'  AND
      'there is at least one applicable rule in the INTRODUCTION RULES set';
    DO
      'chose an applicable rule' r=<L,R> 'in the INTRODUCTION RULES set';
      'chose an occurrence isomorphism' i 'for the application of' r;
      G := (G \ i(L\R)) + i(R\L)
    DONE;

    WHILE G <> 'desired graph'  AND
      'there is at least one applicable rule in the ANALYSIS RULES set';
    DO
      'chose an applicable rule' r=<L,R> 'in the ANALYSIS RULES set';
      'chose an occurrence' i 'for the application of' r;
      G := (G \ i(L\R)) + i(R\L)
    DONE;

    WHILE G <> 'desired graph'  AND
      'there is at least one applicable rule in the DELEGATION RULES set';
    DO
      'chose an applicable rule' r=<L,R> 'in the DELEGATION RULES set';
      'chose an occurrence' i 'for the application of r';
      G := (G \ i(L\R)) + i(R\L)
    DONE

  UNTIL G = 'desired graph' OR 'no applicable rules left';
  IF G = 'desired graph' THEN RETURN(G)
                         ELSE FAIL
END
```

Figure 2.3: Enhanced version of the algorithm for the Tropos Graph generation.

$\overset{G\text{-}I^2}{\Rightarrow} \langle \mathcal{N} \cup \{n_6, n_7\}, \mathcal{E} \cup \{n_6 \to n_1, n_7 \to n_1\}\rangle$
$\overset{G\text{-}DEC\text{-}OR}{\Rightarrow} \langle \mathcal{N}, \mathcal{E} \cup \{n_6 \to n_2, n_7 \to n_2\}\rangle$
$\overset{G\text{-}I^2}{\Rightarrow} \langle \mathcal{N} \cup \{n_8, n_9\}, \mathcal{E}\rangle$
$\overset{G\text{-}P^2}{\Rightarrow} \langle \mathcal{N}, \mathcal{E} \cup \{n_8 \to n_7, n_9 \to n_7\}\rangle \overset{G\text{-}I}{\Rightarrow} \langle \mathcal{N} \cup \{n_{10}\}, \mathcal{E}\rangle$
$\overset{G\text{-}P}{\Rightarrow} \langle \mathcal{N}, \mathcal{E} \cup \{n_{10} \to n_5\}\rangle \overset{G\text{-}I^2}{\Rightarrow} \langle \mathcal{N} \cup \{n_{11}, n_{12}\}, \mathcal{E}\rangle$
$\overset{G\text{-}DEC\text{-}OR^2}{\Rightarrow} \langle \mathcal{N}, \mathcal{E} \cup \{n_{11} \to n_{10}, n_{12} \to n_{10}\}\rangle$

DELEGATION RULES:

$\overset{G\text{-}DEL}{\Rightarrow} \langle \mathcal{N}, \{n_2 \to n_1, n_3 \to n_1, n_5 \to n_4, n_6 \to n_1,$
$n_7 \to n_1, n_6 \to n_2, n_7 \to n_2, n_8 \to n_1,$
$\mathbf{n_1} \to \mathbf{n_9}, \mathbf{n_9} \to \mathbf{n_2}, \mathbf{n_9} \to \mathbf{n_2}, n_8 \to n_7, n_9 \to n_7, n_{10} \to n_4,$
$n_{10} \to n_5, n_{11} \to n_4, n_{12} \to n_4,$
$n_{11} \to n_{10}, n_{12} \to n_{10}\}\rangle$
$\overset{SG\text{-}DEL}{\Rightarrow} \langle \mathcal{N}, \{n_2 \to n_1, \mathbf{n_1} \to \mathbf{n_3}, \mathbf{n_3} \to \mathbf{n_4}, \mathbf{n_3} \to \mathbf{n_4},$
$n_5 \to n_4, n_6 \to n_1, n_7 \to n_1, n_6 \to n_2,$
$n_7 \to n_2,$
$n_8 \to n_1, n_1 \to n_9, n_9 \to n_2, n_9 \to n_2, n_8 \to n_7, n_9 \to n_7, n_{10} \to n_4, n_{10} \to n_5,$
$n_{11} \to n_4, n_{12} \to n_4, n_{11} \to n_{10}, n_{12} \to n_{10}\}\rangle$[12]

END of First LOOP;

As foreseen, after the delegation transformations it may happens that some further analysis can be done, in particular, from the point of view of the actor PAT (node n_4).

Therefore, the algorithm needs to reenter in the main loop.

Second LOOP:

INTRODUCTION RULES: nothing applies;

ANALYSIS RULES:

$\overset{G\text{-}DEC\text{-}OR}{\Rightarrow} \langle \mathcal{N}, \mathcal{E} \cup \{n_{11} \to n_9, n_{12} \to n_9\}\rangle$
$\overset{SG\text{-}I}{\Rightarrow} \langle \mathcal{N} \cup \{n_{13}\}, \mathcal{E} \cup \{n_{13} \to n_4\}\rangle$

$\mathcal{E}$; $\mathcal{N}$ increases always monotonically, thus, only new nodes will be evidenced; $\mathcal{E}$ may be modified non-monotonically, for the effect of some rules (like delegation): in this case the new set will be fully listed. Also, to skip some steps, the notation $\overset{r^2}{\Rightarrow} r$ will be used to denote the application of the rule r twice.

[12]The (soft)goal delegation implies the replacement of the assignment edge (namely, $n_9 \to n_1$ and $n_3 \to n_1$ in the two cases above), with a corresponding chain of the type $actor \to dependum \to actor$, evidenced here in bold.

$\overset{SG\text{-}C\text{-}SG(+)}{\Rightarrow} \langle \mathcal{N}, \mathcal{E} \cup \{n_{13} \rightarrow n_3\}\rangle$
$\overset{SG\text{-}I}{\Rightarrow} \langle \mathcal{N} \cup \{n_{14}\}, \mathcal{E} \cup \{n_{14} \rightarrow n_4\}\rangle$
$\overset{SG\text{-}GEN}{\Rightarrow} \langle \mathcal{N}, \mathcal{E} \cup \{n_{14} \rightarrow n_{13}\}\rangle$
$\overset{SG\text{-}C\text{-}SG(+)}{\Rightarrow} \langle \mathcal{N}, \mathcal{E} \cup \{n_{10} \rightarrow n_{14}\}\rangle$

DELEGATION RULES: nothing applies;

END of Second LOOP

END of REPEAT

END.

It is interesting to notice that, accordingly with the algorithm, before introducing new nodes in the context of an actor by delegation —that, as in the example, may allow for further analysis— all the currently possible analysis must be completed. This is not the most natural way to proceed: in our example, in order to allow for a complete analysis from the point of view of n_4 (`PAT`), it could be more convenient to interleave the analyses of the nodes assigned to n_1 (`Citizen`) and n_4 with the two applications of delegation rules. Also, it is the case to notice that while one of the delegated dependums, n_3 (`tax well spent`), was available just after the introduction phase, the other, n_9 (`system available`), was produced by the analysis of node n_1 . This suggests that switching the order of the analysis and delegation phases would not be a solution to the problem.

Furthermore, during the analysis phases in the first loop, some introduction rules are used. As mentioned previously, they are strictly functional to the analysis steps: they introduce the nodes necessary to perform the analysis itself. This way of proceeding (introduction by need) corresponds to a Top-Down approach: when new nodes are needed to satisfy some "Top" dependum, more specific nodes are introduced. But introduced nodes may as well be used in a Bottom-Up fashion, when, during later steps (see, e.g., the analysis performed in the second loop) they are recognized to be useful also for dependums different from those for which they had been initially introduced. In other examples, more realistic and, thus, dramatically more rich of initially introduced elements, there are much more chances to apply a Bottom-Up analysis since the first steps. Nevertheless, also in these cases more Bottom-Up analyses can emerge after some delegations.

In our example, not much emphasis is given to the process of acquiring the requirements from the stakeholders. It is simply assumed that the engineer is able to perform the diagram development (including application of introduction rules), given an initial knowledge on the domain. In practice this is not the case, and the activity performed by the requirement engineer may require at a certain point (e.g., after the outcome of the

application of some analysis rules, or once a point is reached in which no further rules are applicable, but the resulting diagrams are yet not satisfactory) to acquire new knowledge on the domain, that is, for example, either new documents or new interviews with the stakeholders. Of course, this step can provide insight for the introduction of new nodes in the diagram. Also in this case, it could be the case that these introductions are better not to be delayed until after the strict execution of the main loop.

```
BEGIN
  'initialize graph' G (* in general empty *)
  REPEAT
    'update' agenda; 'sort' agenda;
    <<L,R>,i> := POP(agenda);
    G := (G \ i(L\R)) UNION i(R\L)
  UNTIL G = 'desired graph'  OR  agenda = EMPTY;
  IF G = 'desired graph' THEN RETURN(G)
                         ELSE FAIL
END
```

Figure 2.4: The final version of the algorithm.

For all these reasons, we propose, in Figure 2.4, the final version of the algorithm. It is based on an agenda of applicable rules, which not only allows us to sort rules in categories, but also to manage exceptions in a more flexible way.

In this case the crucial point corresponding to the non deterministic choices is the `'sort'` of the agenda. Updating the agenda, in fact, simply corresponds to adding (or in some cases removing) pairs `<r,i>` (with `r=<L,R>`) to the agenda, accordingly with the given definition of the applicable rule. Differently, the `'sort'` (here unspecified) subroutine must provide for the control over the non-deterministic part of the algorithm, and corresponds to the heuristic part of the algorithm. For example, as a special case, we can reproduce to the previous algorithm, simply by requiring that the rules of the different categories are clustered together in the agenda. But the interesting aspect is that also other criteria can be easily introduced, as, for example, putting first all the rule instances involving a particular actor[13] or including very informal heuristics, such as, e.g., move forth or backward dependum introduction rules accordingly to the kind of actor the dependum is assigned to. In fact, empirical evidence and experience on the domain under analysis may suggest that it could be easier to decide upon the introduction of a new goal of a particular actor over that another, based on the fact the later could require more time or money for the knowledge acquisition steps.[14] There may also be good reasons to keep delegation rules at the bottom of the agenda, for example because it may be judged that a delegation may (riskly) require to revise the already analyzed point of views of other

[13] In our example, using this kind of priority for n_1 would generate an execution track not requiring to reanalyze n_4 goals and softgoals, that is the problem in the execution discussed above.

[14] Consider for example the difference in interviewing an "easily reachable" stakeholder, like a simple employee, instead of interviewing a top-manager of the analyzed organization.

actors. Also, as a final remark, internal analyses, that reduce delegation to a minimum, can be preferred because they are easier to parallelize.

Chapter 3

Transformation of Requirements through Graph Rewriting

3.1 Background

3.1.1 Tropos methodology

As we have seen before Tropos [BGG+03] is an AO methodology for building distributed software systems which provides a visual modeling language. Visual modeling is used for the analysis of the application domain as well as for the requirements, the architectural, and the detailed design of the system-to-be. Among the basic concepts provided by the language: the concept of *actor* for modeling entities which have strategic goals and intentionality, such as a physical agent, a role in an organization or a component in the system-to-be; the concept of *goal* for representing the strategic interests of an actor; the concept of *dependency* between two actors which indicates that an actor depends on another in order to achieve a goal, execute a plan, or exploit a resource. The syntax of the modeling language has been defined through a metamodel, described in [BGG+03]. Tropos offers a set of analysis techniques, such as *and/or decomposition* of goals, *means-end* analysis and *contribution* analysis and a set of diagrams for visualizing model properties. The diagram depicted in Fig. 3.1 gives a graphical representation of goal analysis in a model which includes two actors: the actor "Advisor" and the actor "Producer" representing two stakeholders of the application domain of interest[1]. The actor "Advisor" depends on the actor "Producer" for the achievement of the goal to acquire "orchards data". The dashed balloon includes the analysis of the goal "manage pheromone trap plant" conducted from the point of view of the actor "Advisor". Examples of *and/or decomposition* as well as *contribution* analysis are shown.

The Tropos diagram contains also implicit information about creation and fulfillment

[1]The domain is that of Integrated Production in Agriculture. The example is taken from [PS03].

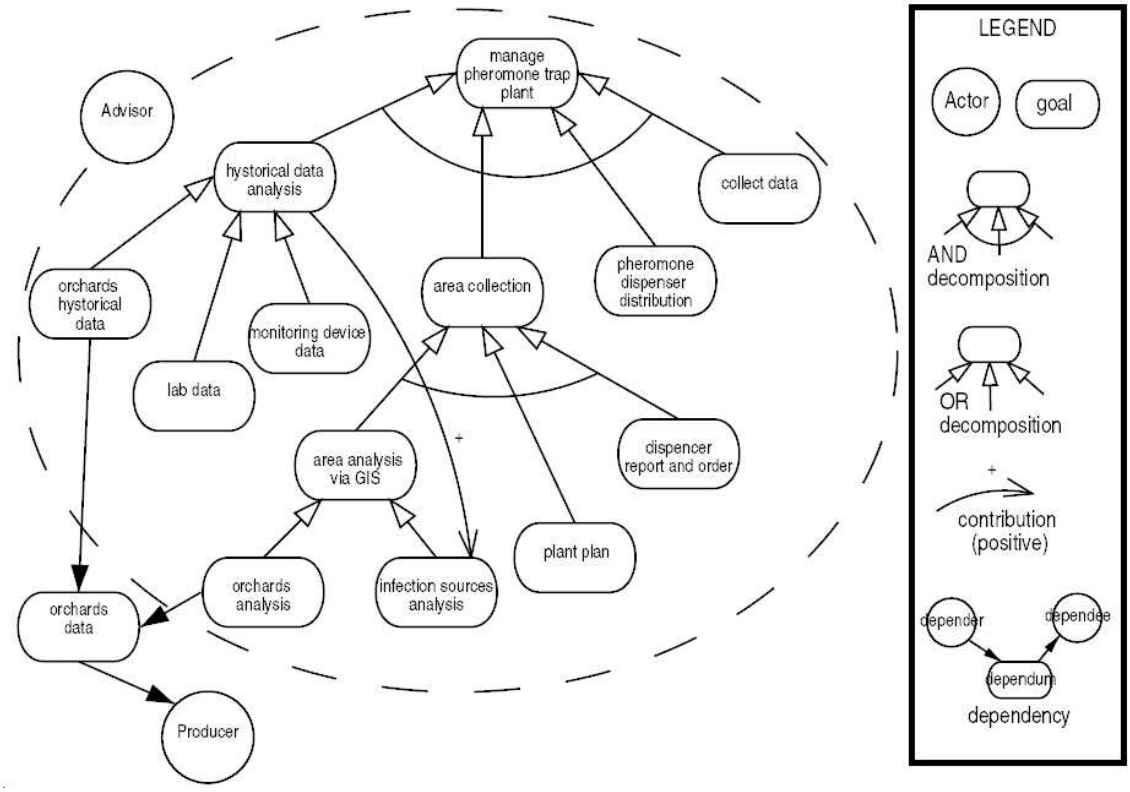


Figure 3.1: An example of Tropos actor and goal diagram.

relations between model's entities. For example, a subgoal can be created only after the creation of an upper level goal and a decomposed goal can be fulfilled only if all the subgoals (*and* decomposition) or at least one subgoal (*or* decomposition) is fulfilled.

The diagram in Fig. 3.2 depicts a frame sequence which expresses a desired behavior of the system modeled in Fig. 3.1. Specifying the frame sequence, an analyst would like to validate the previously specified model with respect to a specific scenario, that is to ensure that the goal "historical data analysis" of the actor instance "a1" (of type "Advisor") can be fulfilled by means of the actor instance "a2" (of type "Producer") who commits herself for the satisfaction of the "a1"'s goal of acquiring "orchards data".

Further information on Tropos and on its application to different case-studies can be found in [Tro].

3.1.2 Supporting Tropos visual modeling

Here we focus on a set of specific issues related to the development of a CASE tool which supports the analyst in the process of building and revising a visual model in Tropos.

The output of a visual modeling process should be a model which is correct with

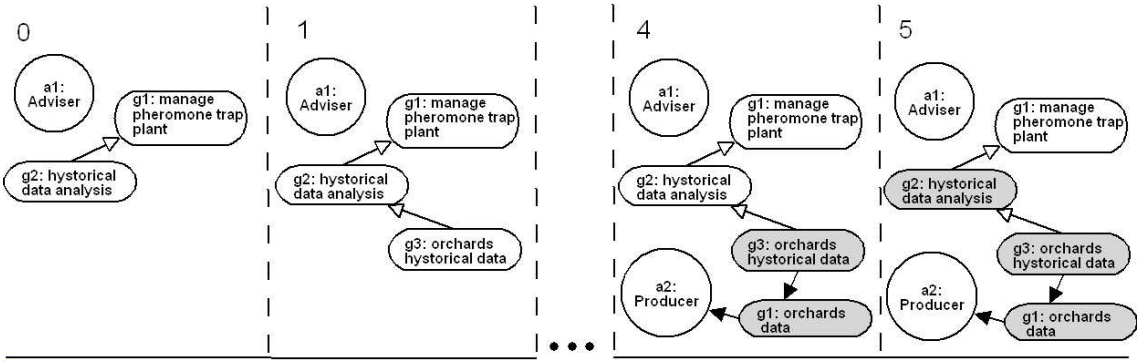


Figure 3.2: An example of frame sequence. Frame 2 to 3 have been omitted. A gray goal is a fulfilled goal. Going from frame 4 to frame 5 we can notice how goal fulfillment propagates from the goal in the dependency towards the original goal, along the goal decomposition chain.

respect to the syntax of the modeling language. However this process goes through the specification of partially correct models. An example could be a creation of a goal without an actor owning it (while according to the language every goal belongs to an actor). Such model should be considered as non completed and the analyst should be warned about this. Moreover, the analyst should be guided towards possible model completions. For example when the analyst starts decomposing a certain goal, the tool should highlight all the goals that can be part of the decomposition. The analyst should also be informed why a building step is forbidden. For instance, if the analyst attempts to create an actor with an already existing name, a clear message should point out the error.

Building a model instance can be helpful for the analyst, for instance, in order to check if the model includes a specific scenario (or behavior). Supporting the analyst in building a consistent instance of a model means that in case she wants to mark ”orchards historical data” in frame 1 of Fig. 3.2 as fulfilled (colored goals) then the tool should warn that it is impossible because the goal should be delegated to the ”Producer” actor which should fulfill it then (e.g. the correct instance is the one depicted in frame 5 of Fig. 3.2).

The tool should be easily adaptable with respect to language variants, such as using only a subset of the modeling language or an extended language where new elements or syntax requirements have been added (for example a new type of decomposition links in addition to *and/or*).

The tool should support also automatic restructuring of the model such as the application of different patterns and refactoring. For instance, if the analyst changes the actor and goal diagram depicted in Fig. 3.1, then the associated instance diagrams should be consistently updated. An example can be a change of a ”historical data analysis” decomposition type from *or* to *and*, which means that in order to fulfil the goal all the subgoals should be fulfilled. So the frame 5 would be no more valid and would require the creation and fulfillment of another subgoal. Another example of model restructuring can be

the movement of a goal and all its subgoals from one actor to another. More generally, restructuring process should guarantee consistency between different views of resulting model.

3.1.3 GR techniques and tools

For solving the problems described above we will use GR. In particular, we consider promising to exploit attributed GR which allow to keep some relevant diagram information in terms of a set of attributes attached to nodes and edges of a graph representing the model. Moreover, the rewriting system should support complex conditions on rules application such as Negative Application Conditions (NACs) and conditions on attributes.

Preliminary evaluation of our ideas have been conducted using a system which implements GR techniques called AGG (Attributed Graph Grammar System) [ERT99]. AGG is a rule based visual language written in java supporting an algebraic approach to graph rewriting. It provides flexible attributed graphs and generic application conditions including NACs and attribute conditions.

3.2 The approach

We describe our approach along three basic steps: in Sec. 3.2.1 we describe a first step in applying GR for guiding the analyst in designing valid models; in Sec. 3.2.2 we address the problem of supporting instance building; in Sec. 3.2.3 we point out techniques needed to provide restructuring/updating capabilities.

3.2.1 Graph rewriting rules for Tropos modeling

A Tropos model can be represented as a graph whose nodes represent entities like actors, goals, resources, plans, and whose edges represent the relationships that can be defined among these entities, such as dependencies and decomposition relationships. Both nodes and edges have attributes which are used to complete the entity specification. It is hence very reasonable to represent the syntax of Tropos using a set of rules that allow for generating all valid Tropos models.

We implemented these GR rules with AGG. The syntax requirements are expressed as positive application conditions (left side of the rules), NACs and conditions on attributes. The graph representation of a Tropos model used by AGG makes explicit some relationships between entities that are implicit in Tropos diagrams, such as the one depicted in Fig. 3.1, by means of additional edges/nodes or attributes. For instance, the fact that a goal belongs to an actor (the goal is inside of a dashed balloon attached to the actor in Fig. 3.1) is expressed with an edge *has* from the actor node to the goal node. Different types of

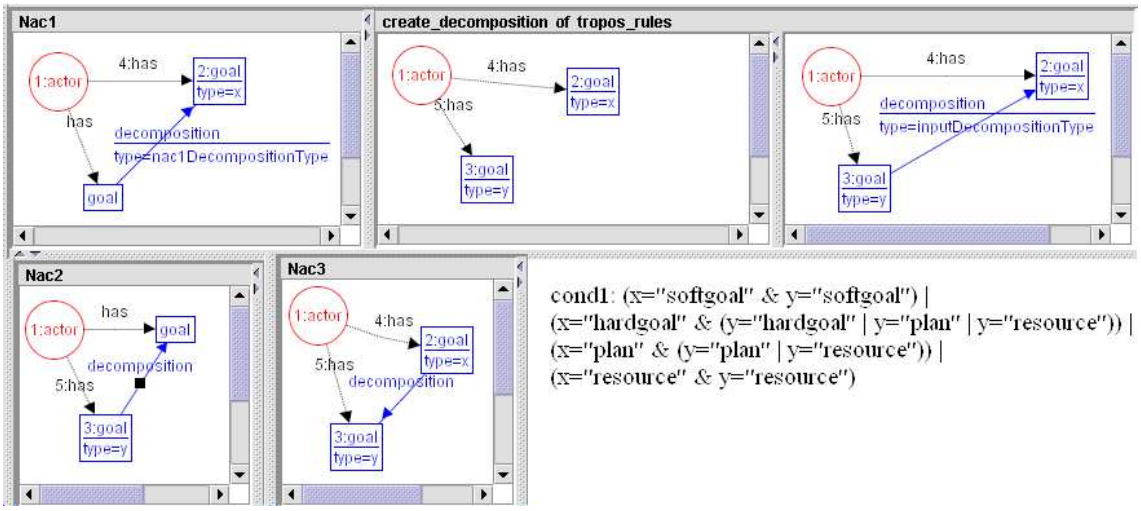


Figure 3.3: An example of rule implemented in AGG. The rule specifies the creation of a decomposition link.

intentional entity (*hardgoal, softgoal, plan, resource*) are represented as attributes of node of type *goal*. This allows to reduce considerably the number of the rewriting rules. All the additional information, like names and parts of formal specification, associated to a node are represented as attributes as well.

Fig. 3.3 contains an example from the rule-set for building a Tropos model. The example is going to be used for the explanation of how we ensure some syntax requirements using GR rules. The input for the rule is the type of the decomposition link (*or/and*). The left side of the rule (positive application condition) guaranties the application of the rule only to two goals belonging to the same actor. The NACs in the rule ensure that we will have the same type of decomposition link from the same root goal and that a decomposition link between the two goals has not been already created. Moreover, it forbids to have a goal that is a child of two different goal decompositions. The condition on attributes "cond1" controls that the source and the target in a decomposition are compatible, according to the Tropos language syntax given in [BGG+03].

Among the advantages of adopting this approach to address the questions posed in Sec. 3.1.2 we'd like to mention the following:

- Flexibility of the modeling system. Let's suppose we change some syntax requirement or introduce new language constructs. In this case we have to change or add NACs to the rules or add new rules, which is a limited effort.

- Guiding the modeling process. We can exploit the functionalities provided by AGG such as suggesting possible rule mapping completions, for instance to highlight possible completion objects. In the case of the rule for creating a decomposition link, the analyst can select the goal to be decomposed and the tool can point out all

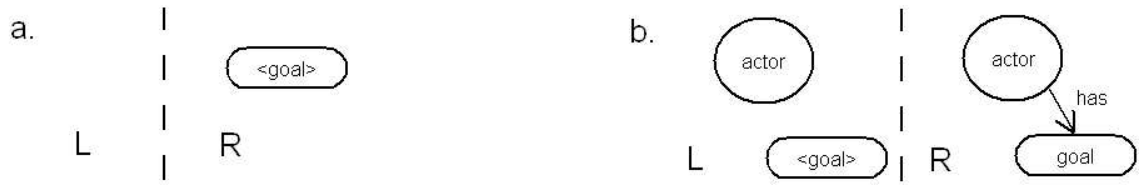


Figure 3.4: The example of rules for managing nonterminal elements.

the possible subgoals that are valid completions for the rule.

- Build partially correct models. In order to allow for partially correct models (e.g. models with goals not associated to actors) we can extend the set of types for building a Tropos models to include terminal and nonterminal types. The model is considered to be partially correct if the graph contains nonterminal typed elements. Only graphs containing terminal elements represent correct models. An example of rule involving nonterminal elements is shown in the Fig. 3.4. The rule "a." creates a goal without actor. So, we are using a nonterminal type for such kind of goal: $< goal >$ which is an incorrect entity according to the Tropos syntax because it doesn't belong to an actor. A $< goal >$ can not be further analyzed, e.g. it cannot be decomposed. The second rule "b." allows to assign an actor to the nonterminal goal. Here the nonterminal element $< goal >$ is replaced with the terminal element *goal*.

3.2.2 Building Tropos instances and frame sequences

A Tropos model represents domain actors, goals and dependencies and it may include, as well, instances of those entities. The process of building instances requires an additional set of rules that can be applied to the same model. The set of rules for creating instances has to take the fulfillment/creation dependency information (described in Sec. 3.1.1) into account as rules conditions.

Fig. 3.5 depicts an example of rules of this kind. There are two new types of links introduced in the rule: *instance of* link means that the object is an instance of a Tropos diagram object, the link *belongs* means that an actor instance (and thus all its goal instances) belong to an instance diagram.

The rule Fig. 3.5a creates a new node of type *instance diagram* and thus a new instance diagram. The rule in Fig. 3.5b creates an instance of goal. The NAC for this rule ensures that the goal is not a subgoal. (The rule for a subgoal creation has not been described in the Fig. 3.5.) The rule in Fig. 3.5c changes the status of an upper level goal to "fulfilled" only if all the subgoals are fulfilled (*and* decomposition) or at least one subgoal is fulfilled (*or* decomposition).

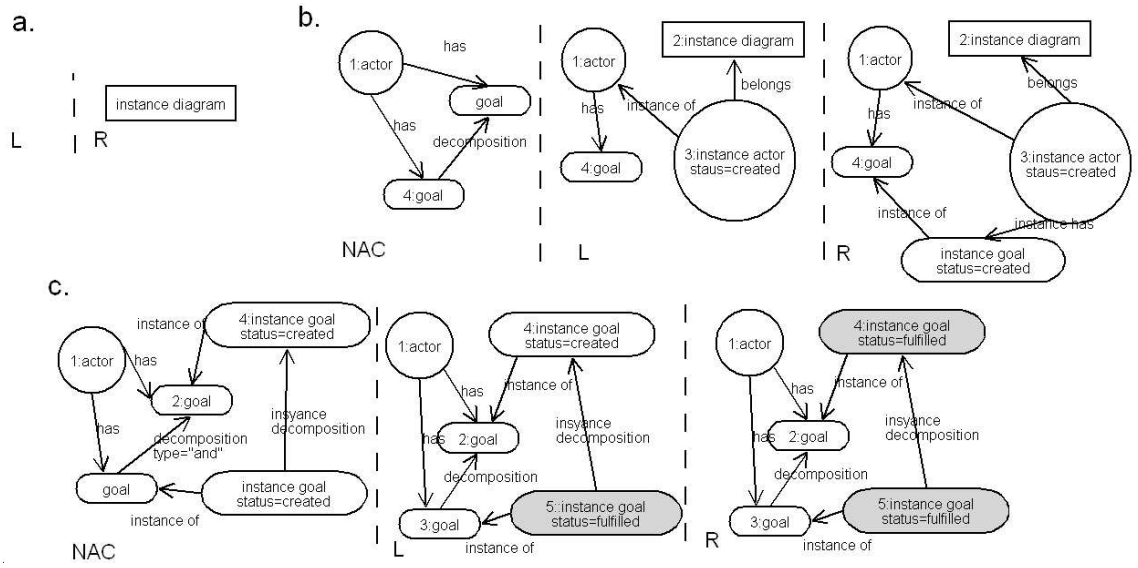


Figure 3.5: The example of the rules for instance creation: a) the rule creating an instance diagram label; b) the rule for creating a subgoal; c) the rule marking fulfillment of decomposed goal.

We remark that an instance diagram is a snapshot of a frame sequence (see Fig. 3.2). We can reproduce a frame sequence for it by reapplying the set of rules for the instance building. Every frame corresponds to a step of the instance building process which is performed by the analyst starting from the rule *instance diagram* as a first, empty frame.

The main advantage of the approach here is that it guaranties to build instances within a Tropos model. Using the set of rules for the instances we can create any number of instances for the Tropos diagram. And, so that all of them are guarantied to be compatible with the model itself.

3.2.3 Derivation transformations

The GR rules described above include only rules for creating new elements. While additional rules can be added for some deletion and changing activities it looks like several of these activities can not be done with only one graph rewriting rule. There are two reasons why: first a deletion step may require many preconditions (e.g., we can delete an actor only after having deleted all the goals belonging to it); second, a rewriting step may impact to a large subset of the model (e.g., movement of a goal from an actor to another involves movement of all the dependent subgoals, Fig. 3.6). For such kind of diagram modifications we foresee a derivation transformation approach as the one described in [HM02, HM00]. This will provide a general, semantically found solution to the problem.

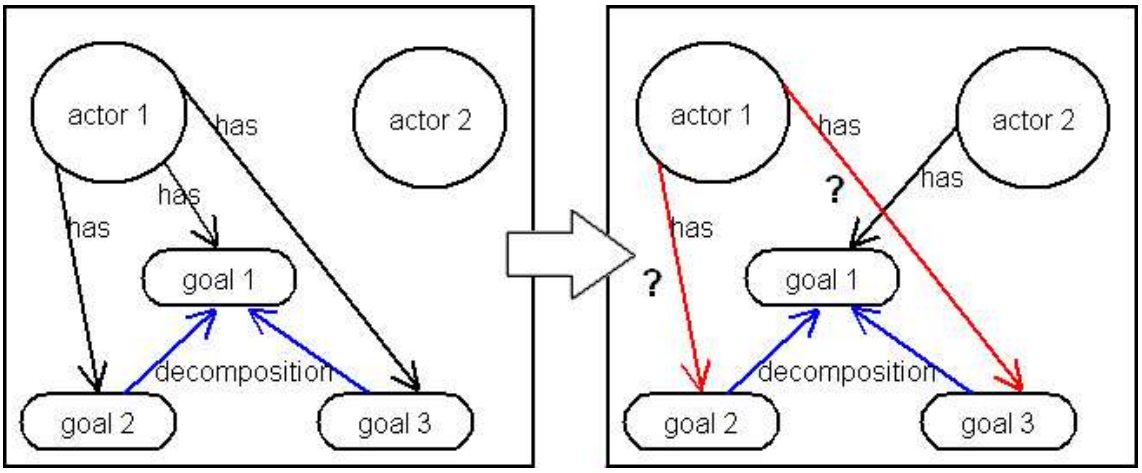


Figure 3.6: The movement of a goal to another actor.

For the moment, we are experimenting with a more naive approach which is based on the idea of keeping and saving the derivation history and of applying external, hand-written algorithms to represent specific derivation transformations. For example such an algorithm should support goal movement from one actor to the other. It should change the rule for creation of the goal replacing the original actor with the desired one, it should check all the dependency links and all the subgoals, propagating then the same rule transformation to the subgoals. All the other links, except decomposition, should be destroyed. The new derivation can be reapplied resulting into the changed diagram. Analogous updating will be done for all the instance diagrams. The same technique can be used for other type of restructuring in a model.

Some of the derivation transformation algorithms can require the analyst to input his/her decision. For example a decision on what the analyst is going to do with a specific type of link in the diagram while moving a goal from one actor to another, or on how the analyst would like to rename a goal.

3.3 Related work

In the field of GR for visual modeling there are different directions, and among them OO modeling with UML is gaining a lot of attention. In particular, several works address issues of consistency and semantics of the different types of UML diagrams. Some works on consistency of UML class and sequence diagrams use attributed GR for checking of consistency of the two types of diagrams [TE00]. In [GZK03] an approach is described for an integrated graph based semantics for UML diagrams such as class, object and state diagrams. The idea of the paper is that the analyst associates graph rewriting rule to the operations of a class in a class diagram. Using these hand-written rules and other

rules generated from statechart diagrams, it is possible to check consistency of object, sequence, and collaboration diagrams. Also in our approach we are using GR as a basis for consistency between static and dynamic views of a Tropos model. However, our approach does not require the analyst to write rules by hand, since a predefined set of rules is sufficient to support instantiation taking into account Tropos language semantics.

Among the available tools that implement GR and that offer also advanced editing functionalities we considered GenGEd or DiaGen both described in [BGMS99]. We preferred to use the AGG tool since our focus is on providing a graph transformation engine. This engine shall then be integrated in a CASE tool which provides a broader set of functionalities, and that is being developed in our research group.

Chapter 4

Transformation of Security Requirements through Planning

4.1 Secure Tropos

Secure Tropos [GMMZ05a] is a RE methodology for modeling and analyzing functional and security requirements, extending the Tropos methodology [CKM02]. This methodology is tailored to describe both the system-to-be and its organizational environment starting with early phases of the system development process. The main advantage of this approach is that one can capture not only the *what* or the *how*, but also the *why* a security mechanism should be included in the system design. In particular, Secure Tropos deals with business-level (as opposed to low-level) security requirements. The focus of such requirements includes, but is not limited to, how to build trust among different partners in a virtual organization and trust management. Although their name does *not* mention security, they are generally regarded as part of the overall security framework.

Secure Tropos uses the concepts of actor, goal, task, resource and social relations for defining entitlements, capabilities and responsibilities of actors. An *actor* is an intentional entity that performs actions to achieve goals. A *goal* represents an objective of an actor. A *task* specifies a particular sequence of actions that should be executed for satisfying a goal. A *resource* represents a physical or an informational entity.

Actors' desires, entitlements, capabilities and responsibilities are defined through social relations. In particular, Secure Tropos supports *requesting*, *ownership*, *provisioning*, *trust*, and *delegation*. Requesting identifies desires of actors. Ownership identifies the legitimate owner of a goal, a task or a resource, that has full authority on access and disposition of his possessions. Provisioning identifies actors who have the capabilities to achieve a goal, execute a task or deliver a resource. We demonstrate the use of these concepts through the design of a Medical IS for the payment of medical care.[1]

[1] An extended description of the example is provided in [BMMZ06].

Example 1 *The* Health Care Authority *(*HCA*) is the "owner" of the goal* provide medical care*; that is, it is the only one that can decide who can provide it and through what process. On the other hand,* Patient *wants this goal fulfilled. This goal can be AND-decomposed into two subgoals:* provisioning of medical care *and* payment for medical care. *The* Healthcare Provider *has the capability for the* provisioning of medical care*, but it should wait for authorization from* HCA *before doing it.*

Delegation of execution is used to model situations where an actor (the delegator) delegates the responsibilities to achieve a goal, execute a task, or delivery a resource to another actor (the delegatee) since he does not have the capability to provide one of above by himself. It corresponds to the actual choice of the design. *Trust of execution* represents the belief of an actor (the trustor) that another actor (the trustee) has the capabilities to achieve a goal, execute a task or deliver a resource. Essentially, delegation is an action due to a decision, whereas trust is a mental state driving such decision. Tropos dependency can be defined in terms of trust and delegation [GMMZ05b]. Thus, a Tropos model can be seen as a particular Secure Tropos model. In order to model both functional and security requirements, Secure Tropos introduces also relations involving permission. *Delegation of permission* is used when in the domain of analysis there is a formal passage of authority (e.g. a signed piece of paper, a digital credential, etc.). Essentially, this relation is used to model scenarios where an actor authorizes another actor to achieve a goal, execute a task, or deliver a resource. It corresponds to the actual choice of the design. *Trust of permission* represents the belief of an actor that another actor will not misuse the goal, task or resource.

Example 2 *The* HCA *must choose between different providers for the welfare management for executives of a public institution. Indeed, since they have a special private-law contract, they can qualify for both the* INPDAP *and* INPDAI[2] *welfare schemes. The* INPDAP *scheme requires that the* Patient *partially pays for medical care (with a ticket) and the main cost is directly covered by the* HCA*. On the contrary, the* INPDAI *scheme requires that the* Patient *pays in advance the full cost of medical care and then gets reimbursed. Once an institution has decided the payment scheme, this will be part of the requirements to be passed onto the next stages of system development. Obviously, the choice of the alternative may have significant impacts on other parts of the design.*

Figure 4.1 summarizes Examples 1 and 2 inTransformation of Security Requirements through Planning terms of a Secure Tropos model. In this diagram, actors are represented as circles and goals as ovals. Labels **O**, **P** and **R** are used for representing ownership, provisioning and requesting relations, respectively. Finally, we represent trust of permission and trust of execution relationships as edges respectively labelled **Tp** and **Te**.

[2]INPDAP (Istituto Nazionale di Previdenza per i Dipendenti dell'Amministrazione Pubblica) and INPDAI (Istituto Nazionale di Previdenza per i Dirigenti di Aziende Industriali) are two Italian national welfare institutes.

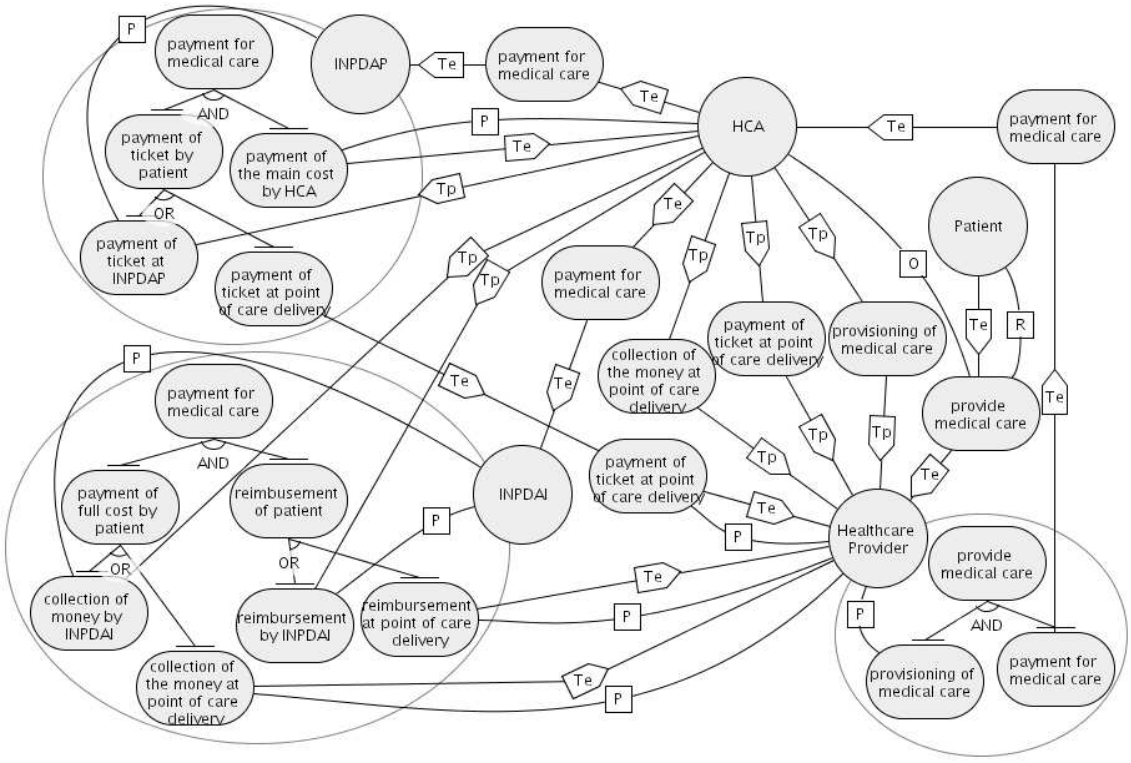


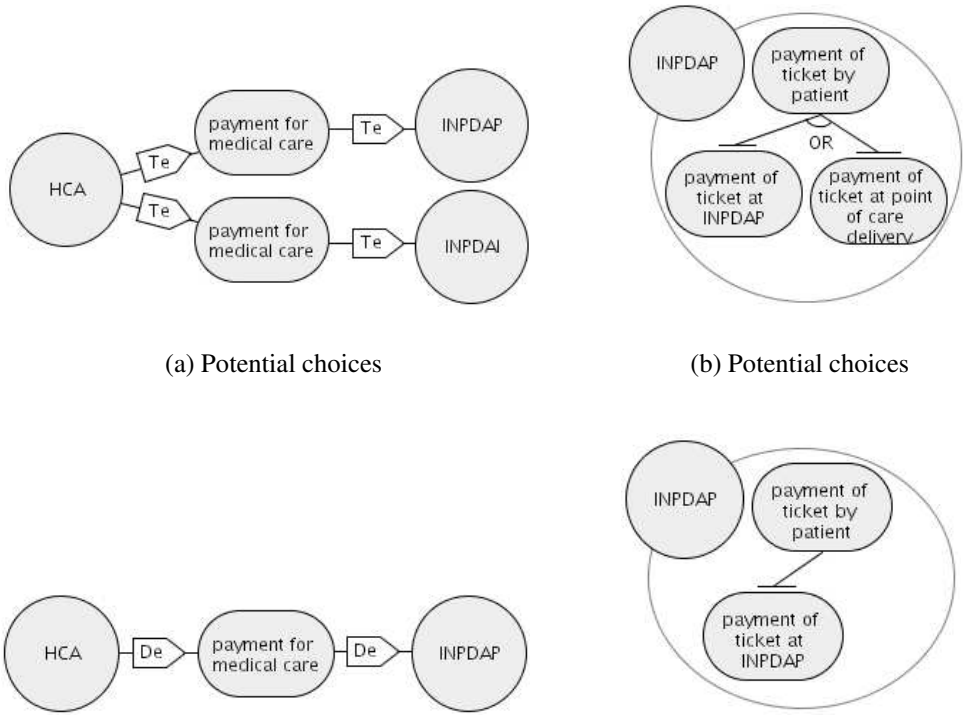
Figure 4.1: Secure Tropos model

Once a stage of the *modeling phase* is concluded, Secure Tropos provides mechanisms for the verification of the model [GMMZ05a]. This means that the design process iterates over the following steps:

- model the system;
- translate the model into a set of clauses (this is done automatically);
- verify whether appropriate design or security patterns are satisfied by the model.

Through this process, we can verify the compliance of the model with desirable properties. For example, it can be checked whether the delegator trusts that the delegatee will achieve a goal, execute a task or deliver a resource (trust of execution), or will use a goal, task or resource correctly (trust of permission). Other desirable properties involve verifying whether an actor who requires a service, is confident that it will be delivered. Furthermore, an owner may wish to delegate permissions to an actor only if the latter actually does need the permission. For example, we want to avoid the possibility of having alternate paths of permission delegations. Secure Tropos provides support for identifying all these situations.

Secure Tropos has been used for modeling and analyzing real and comprehensive case studies where we have identified vulnerabilities affecting the organizational structure of a



(a) Potential choices

(b) Potential choices

(c) Actual choice

(d) Actual choice

Figure 4.2: Design Alternatives

bank and its IT system [MZ06], and verified the compliance to the Italian legislation on Privacy and Data Protection by the University of Trento [MPZ05].

4.2 Design as Planning

So far the automated reasoning capabilities of Secure Tropos are only able to check that subtle errors are not overlooked. This is rather unsatisfactory from the point of view of the designer. Whereas he may have a good understanding of possible alternatives, he may not be sure which is the most appropriate alternative for the case at hand. This is particularly true for delegations of permission that need to comply with complex privacy regulations (see [MPZ05]).

Example 3 *Figures 4.2(a) and 4.2(b) present fragments of Figure 4.1, that point out the potential choices of the design. The requirements engineer has identified trust relations between the* HCA *and* INPDAP *and* INPDAI. *However, when passing the requirements onto the next stage only one alternative has to be selected because that will be the system that is chosen. Figures 4.2(c) and 4.2(d) present the actual choices corresponding to the potential choices presented in Figures 4.2(a) and 4.2(b), respectively.*

Here, we want to support the requirements engineer in the selection of the best alternative by changing the design process as follows:

- Requirements analysis phase
  - Identify system actors along with their desires, capabilities and entitlements, and possible ways of goal decomposition.
  - Define trust relationships among actors both in terms of execution and permission.
- Design phase
  - The space of design alternatives is automatically explored to identify delegation of execution/permission.
  - Depending on the time/importance of the goal the designer may settle for satisficing solutions [Sim69] or ask for an optimal solution.

To support the designer in the process of selecting the best alternative we advocate a planning approach which recently has proved to be applicable in the field of automatic Web service composition [CST03].

The basic idea behind the planning approach is to automatically determine the course of actions (i.e. a plan) needed to achieve a certain goal where an action is a transition rule from one state of the system to another [Wel99, Pee05]. Actions are described in terms of preconditions and effects: if the precondition is true in the current state of the system, then the action is performed. As consequence of the action, the system will be in a new state where the effect of the action is true. Thus, once we have described the initial state of the system, the goal that should be achieved (i.e. the desired final state of the system), and the set of possible actions that actors can perform, the solution of the planning problem is the (not necessarily optimal) sequence of actions that allows the system to reach the desired state from the initial state.

In order to cast the design process as a planning problem, we need to address the following question: *which are the "actions" in a software design?* When drawing the Secure Tropos model, the designer assigns the execution of goals from one actor to another, delegates permission and – last but not least – identifies appropriate goal refinements among selected alternatives. These are the actions to be used by the planner in order to fulfill all initial actor goals.

4.3 Planning Domain

The planning approach requires a specification language to represent the planning domain and the states of the system. Different types of logics could be applied for this purpose,

Goal Properties
$\mathsf{AND\_decomposition_n}(\mathsf{g} : \mathsf{goal}, \mathsf{g_1} : \mathsf{goal}, \ldots, \mathsf{g_n} : \mathsf{goal})$
$\mathsf{OR\_decomposition_n}(\mathsf{g} : \mathsf{goal}, \mathsf{g_1} : \mathsf{goal}, \ldots, \mathsf{g_n} : \mathsf{goal})$
Actor Properties
$\mathsf{provides}(\mathsf{a} : \mathsf{actor}, \mathsf{g} : \mathsf{goal})$
$\mathsf{requests}(\mathsf{a} : \mathsf{actor}, \mathsf{g} : \mathsf{goal})$
$\mathsf{owns}(\mathsf{a} : \mathsf{actor}, \mathsf{g} : \mathsf{goal})$
Actor Relations
$\mathsf{trustexe}(\mathsf{a} : \mathsf{actor}, \mathsf{b} : \mathsf{actor}, \mathsf{g} : \mathsf{goal})$
$\mathsf{trustper}(\mathsf{a} : \mathsf{actor}, \mathsf{b} : \mathsf{actor}, \mathsf{g} : \mathsf{goal})$

Table 4.1: Primitive Predicates.

e.g. first order logic is often used to describe the planning domain with conjunctions of literals[3] specifying the states of the system. We find this representation particularly useful for modeling real case studies. Indeed, when considering security requirements at enterprise level, one must be able to reason both at the class level (e.g. the CEO, the CERT team member, the employee of the HR department) and at the instance level (e.g. John Doe and Mark Doe playing those roles).

The planning domain language should provide support for specifying:

- the initial state of the system,
- the goal of the planning problem,
- the actions that can be performed,
- the axioms of background theory.

Table 4.1 presents the predicates used to describe the *initial state of the system* in terms of actor and goal properties, and social relations among actors. We use

- AND/OR_decomposition to describe the possible decomposition of a goal;
- provides, requests and owns to indicate that an actor has the capabilities to achieve a goal, desires the achievement of a goal, and is the legitimate owner of a goal, respectively;
- trustexe and trustper to represent trust of execution and trust of permission relations, respectively.

[3]Let p be a predicate symbol with arity n, and $t_1, \ldots, t_n$ be its corresponding arguments. $p(t_1, \ldots, t_n)$ is called an *atom*. The expression *literal* denotes an atom or its negation.

Basic Actions
DelegateExecution(a : actor, b : actor, g : goal)
DelegatePermission(a : actor, b : actor, g : goal)
Satisfy(a : actor, g : goal)
AND_Refine$_n$(a : actor, g : goal, g_1 : goal, . . . , g_n : goal)
OR_Refine$_n$(a : actor, g : goal, g_1 : goal, . . . , g_n : goal)
Absence of Trust
Negotiate(a : actor, b : actor, g : goal)
Contract(a : actor, b : actor, g : goal)
DelegateExecution_under_suspicion(a : actor, b : actor, g : goal)
Fulfill(a : actor, g : goal)
Evaluate(a : actor, g : goal)

Table 4.2: Actions.

The desired state of the system (or *goal of the planning problem*) is described through the conjunction of predicates done derived from the requesting relation in the initial state. Essentially, for each request(a,g) we need to derive done(g).

By contrast, an *action* represents an activity to accomplish a goal. We list them in Table 4.2 and define them in terms of preconditions and effects as follows:

Satisfy. The satisfaction of goals is an essential action. Following the definition of goal satisfaction given in [GMMZ05a], we say that an actor satisfies a goal only if the actor wants and is able to achieve the goal, and – last but not least – he is entitled to achieve it. The effect of this action is the fulfillment of the goal.

DelegateExecution. An actor may not have enough capabilities to achieve assigned goals by himself, and so he has to delegate their execution to other actors. We represent this passage of responsibilities through action DelegateExecution. It is performed only if the delegator requires the fulfillment of the goal and trusts that the delegatee will achieve it. Its effect is that the delegator does not worry any more about the fulfillment of this goal after delegating it since he has delegated its execution to a trusted actor. Furthermore, the delegatee takes the responsibility for the fulfillment of the goal and so it becomes a his own desire. Notice that we do not care how the delegatee satisfies the goal (e.g. by his own capabilities or by further delegation). It is up to the delegatee to decide it.

DelegatePermission. In the initial state of the system, only the owner of a goal is entitled to achieve it. However, this does not mean that he wants it or has the capabilities to achieve it. On the contrary, in the system there may be some actors that want that goal and others that can achieve it. Thus, the owner could decide to authorize trusted actors to achieve the goal. The formal passage of authority takes place when the owner issues a certificate that authorizes another actor to achieve the goal.

We represent the act of issuing a permission through action DelegatePermission which is performed only if the delegator has the permission on the goal and trusts that the delegatee will not misuse the goal. The consequence of this action is to grant rights (on the goal) to the delegatee, that, in turn, can re-delegate them to other trusted actors.

AND/OR_Refine. An important aspect of Secure Tropos is goal refinement. In particular, the framework supports two types of refinement: OR_decomposition, which suggests the list of alternative ways to satisfy the goal, and AND-decomposition, which refines the goals into subgoals which all are to be satisfied in order to satisfy the initial goal. We introduce actions AND_Refine and OR_Refine. Essentially, AND_Refine and OR_Refine represent the action of refining a goal along a possible decomposition. An actor refines a goal only if he actually need it. Thus, a precondition of AND_Refine and OR_Refine is that the actor requests the fulfillment of the initial goal. A second precondition determines the way in which the goal is refined. The effect of AND_Refine and OR_Refine is that the actor who refines the goal focuses on the fulfillment of subgoals instead of the fulfillment of the initial goal.

In addition to actions we define *axioms* in the planning domain. These are rules that hold in every state of the system and are used to complete the description of the current state. They are used to propagate actors and goal properties along goal refinement: a goal is satisfied if all its AND-subgoals or at least one of the OR-subgoals are satisfied. Moreover, axioms are used to derive and propagate entitlements. Since the owner is entitled to achieve his goals, execute his tasks and access his resources, we need to propagate actors' entitlements top-down along goal refinement.

4.4 Delegation and Contract

Many business and social studies have emphasized the key role played by trust as a necessary condition for ensuring the success of organizations [Dru90]. Trust is used to build collaboration between humans and organizations since it is a necessary antecedent for cooperation [Axe84]. However, common sense suggests that fully trusted domains are simply idealizations. Actually, many domains require that actors who do not have the capabilities to fulfill their objectives, must delegate the execution of their goals to other actors even if they do not trust the delegatees. Accordingly, much work in recent years has focused on the development of frameworks capable of coping with lack of trust, sometimes by introducing an explicit notion of distrust [GJKL01, GMMZ05b].

The presence (or lack) of trust relations among system actors particularly influences the strategies to achieve a goal [Luh79]. In other words, the selection of actions to fulfill a goal changes depending on the belief of the delegator about the possible behavior of the delegatee. In particular, if the delegator trusts the delegatee, the first is confident that the

latter will fulfill the goal and so he does not need to verify the actions performed by the delegatee. On the contrary, if the delegator does not trust the delegatee, the first wants some form of control on the behavior of the latter.

Different solutions have been proposed to ensure for the delegator the fulfillment of his objectives. A first batch of solutions comes from transaction cost economics and contract theories that view a contract as a basis for trust [KHN05]. This approach assumes that a delegation must occur only in the presence of trust. This implies that the delegator and the delegatee have to reach an agreement before delegating a service. Essentially, the idea is to use a contract to define precisely what the delegatee should do and so establish trust between the delegator and the delegatee. Other theories propose models where effective performance may occur also in the absence of trust [Gal01]. Essentially, they argue that various control mechanisms can ensure the effective fulfillment of actors's objectives.

In this chapter we propose a solution for delegation of execution that borrows ideas from both approaches. The case for delegation of permission is similar. The process of delegating in the absence of trust is composed of two phases: *establishing trust* and *control*. The establishing trust phase consists of a sequence of actions, namely Negotiate and Contract. In Negotiate the parties negotiate the duties and responsibilities accepted by each party after delegation. The postcondition is an informal agreement representing the initial and informal decision of parties to enter into a partnership. During the execution of Contract the parties formalize the agreement established during negotiation. The postcondition of Contract is a trust "under suspicion" relation between the delegator and the delegatee. Once the delegator has delegated the goal and the delegatee has fulfilled the goal, the first wants to verify if the latter has really satisfied his objective. This control is performed using action Evaluation. Its postcondition is the "real" fulfillment of the goal. To support this solution we have introduced some additional actions (last part of Table 4.2) to distinguish the case in which the delegation is based on trust from the case in which the delegator does not trust the delegatee.

Sometimes establishing new trust relations might be more convenient than extending existing trust relations. A technical "side-effect" of our solution is that it is possible to control the length of trusted delegation chains. Essentially, every action has a unit cost. Therefore, refining an action into sub-actions corresponds to increasing the cost associated with the action. In particular, refining the delegation action in absence of trust guarantees that the framework first tries to delegate to trusted actors, but if the delegation chain results too long the system can decide to establish a new trust relation rather than to follow the entire trust chain.

Need-to-know property of a design decision states that the owner of a goal, a task or a resource wants that only the actors who need permission on its possession are authorized to access it. Essentially, only the actor that achieves a goal, executes a task or delivers a resource, and the actors that belong to the delegation of permission chain from the owner to the provider should be entitled to access this goal, task or resource. Thus, we want to obtain a plan where only the actions that contribute to reaching the desired state occur,

Planner	Release	URL
DLV$^{\mathcal{K}}$	2005-02-23	http://www.dbai.tuwien.ac.at/proj/dlv/K/
IPP 4.1	2000-01-05	http://www.informatik.uni-freiburg.de/ koehler/ipp.html
CPT 1.0	2004-11-10	http://www.cril.univ-artois.fr/ vidal/cpt.en.html
SGPLAN	2004-06	http://manip.crhc.uiuc.edu/programs/SGPlan/index.html
SATPLAN	2004-10-19	http://www.cs.washington.edu/homes/kautz/satplan/
LPG-td	2004-06	http://zeus.ing.unibs.it/lpg/

Table 4.3: Comparison among planners.

Requirement \ Planner	DLV$^{\mathcal{K}}$	IPP	CPT	SGPLAN	SATPLAN	LPG-td
1		X	X	X	X	X
2				X	X	X
3	X	X	X			X

Table 4.4: Comparison among planners.

so that if any action is removed from the plan it no longer satisfies the goal of the planning problem. This approach guarantees the absence of alternative paths of permission delegations since a plan does not contain any redundant actions.

4.5 Using the Planner

In the last years many planners have been proposed (Table 4.3). In order to choose one of them we have analyzed the following requirements:

1. The planner should produce solution that satisfy need-to-know property by construction, that is, the planner should not produce redundant plans. Under non-redundant plan we mean that, by deleting an arbitrary action of the plan, the resulting plan is no more a "valid" plan (i.e. it does not allow to reach the desired state from the initial state).

2. The planner should use PDDL (Planning Domain Definition Language) [GHK$^+$98], since it is becoming the "standard" planning language and many research groups work on its implementation. In particular, the planner should use PDDL 2.2 specifications [EH04], since this version support features, such as derived predicates, that are essential for implementing our planning domain.

3. The planner should be available on both Linux and Windows platforms as our previous Secure Tropos reasoning tool works on both.

```
(: action Satisfy
   : parameters (?a − actor ?g − goal)
   : precondition (and
      (provides ?a ?g)
      (requests ?a ?g)
      (has_per ?a ?g))
   : effect (and
      (done ?g)
      not (requests ?a ?g)))
```

(a) Satisfy

```
(: action DelegatePermission
   : parameters (?a ?b − actor ?g − goal)
   : precondition (and
      (trustper ?a ?b ?g)
      (has_per ?a ?g))
   : effect (and
      (has_per ?b ?g)))
```

(b) DelegatePermission

Figure 4.3: Actions' Specification

```
DelegateExecution Pat HP ProvideMC
AND_Refine HP ProvideMC ProvisioningMC PaymentMC
DelegatePermission HCA HP ProvisioningMC
Satisfy HP ProvisioningMC
DelegateExecution HP HCA PaymentMC
DelegateExecution HCA INPDAP PaymentMC
AND_Refine INPDAP PaymentMC PaymentTicket PaymentHCA
DelegateExecution HCA INPDAP PaymentHCA
Satisfy HCA PaymentHCA
OR_Refine INPDAP PaymentTicket PaymentTicketINPDAP PaymentTicketHP
DelegatePermission HCA INPDAP PaymentTicketINPDAP
Satisfy INPDAP PaymentTicketINPDAP
```

Figure 4.4: The optimal solution

Table 4.4 presents a comparison among the planners we have considered with respect to above requirements. Based on such requirements, we have chosen LPG-td, a fully automated system for solving planning problems, supporting PDDL 2.2. Figure 4.3 shows the specification of actions Satisfy and DelegatePermission in PDDL 2.2.

We have applied our approach to the Medical IS presented in Figure 4.1. The desired state of the system is obviously one where the patient gets medical care. The PDDL 2.2 specification of the planning problem is given in [BMMZ06].

Figure 4.4 shows the optimal solution (i.e. the plan composed of the fewer number of actions than any other plan) proposed by LPG-td. However, this was not the first choice of the planner. Before selecting this plan, the planner proposed other two sub-optimal alternatives (see [BMMZ06] for a discussion). It is interesting to see that the planner has first provided a solution with INPDAP, then a solution with INPDAI, and then, finally, a revised solution with INPDAP. A number of other experiments were conduced to test the

scalability of our approach. The results are reported in [BMMZ06].

4.6 Related Work

In recent years many efforts have addressed the integration of security with the system development process, in particular during early requirements analysis. In this setting, many researchers have recognized trust as an important aspect of this process since trust influences the specification of security and privacy policies. However, very few requirements engineering methodologies introduce trust concerns during the system development process. Yu et al. [YL01] model trust by using the concept of softgoal, i.e. goal having no clear definition for deciding whether it is satisfied or not. However, this approach considers trust as a separate concept from security and does not provide a complete framework to consider security and trust throughout the development process. Haley et al. [HLMN06] propose to use trust assumptions, problem frames, and threat descriptions to aid requirements engineers to define and analyze security requirements, and to document the decisions made during the process.

Other approach focus on security requirements without taking into account trust aspect. van Lamsweerde et al introduce the notion of antigoals for representing the goals of attackers [vBDJ03]. McDermott et al. define abuse case model [MF99] to specify the interactions among actors, whose the results are harmful to some actors. Similarly, Sindre et al. define the concept of a misuse case [SO05], the inverse of a use case, which describes a function that the system should block.

Model Driven Architecture (MDA) approach [Gro01], proposed by Object Management Group, is a framework for defining software design methodologies. Its central focus is on the model transformation, for instance from the platform-independent model of the system to platform-specific models used for implementation purposes. Models are usually described in UML, and the transformation is performed in accordance with the set of rules, called mapping. Transformation could be manual, or automatic, or mixed. Among the proposals on automating a software design process the one of Gamma et al. on design patterns [GHJV95] has been widely accepted. A design pattern is a solution (commonly observed from practice) to the certain problem in the certain context, so it may be thought as a problem-context-solution triple. Several design patterns can be combined to form a solution. Notice that it is still the designer who makes the key decision on what pattern to apply to the given situation.

The field of AI planning have been making advances during the last decades, and has found a number of applications (robotics, process planning, autonomous agents, Web services, etc.). There two basic approaches to the solution of planning problems [Wel99]. One is graph-based planning algorithms [BF97] in which a compact structure called a Planning Graph is constructed and analyzed. While in the other approach [KS92] the planning problem is transformed into a SAT problem and a SAT solver is used. An ap-

plication of the planning approach to requirements engineering is proposed by Gans et al. [GJKL01]. Essentially, they propose to map trust, confidence and distrust described in terms of i* models [YL01] to delegation patterns in a workflow model. Their approach is inspired by and implemented in ConGolog [dGLL00], a logic-based planning language. In this setting, tasks are implemented as ConGolog procedures where preconditions correspond to conditionals and interrupts. Also monitors are mapped into ConGolog procedures. They run concurrently to the other agent tasks waiting for some events such as task completion and certificate expiration. However, their focus is on modeling and reasoning about trust in social networks, rather than on secure design.

Chapter 5

Conclusion

As presented in [BPG+02], the Tropos analysis process can be defined in terms of transformation applications, and in particular, transformations that can be applied for refining an initial Tropos model to a final one, working incrementally. In this deliverable we have focused first of all on the redefinition of the transformation system for Tropos early requirements analysis in terms of a Graph Transformation System. The formalization provides the necessary machinery to perform precise inspections of the process of early requirements analysis, and allows us to distinguish among different strategies for the execution of the process. We have also introduced some execution algorithms with the help of an example, and finally, we have discussed some preliminary observations on different control strategies.

The work and the considerations presented in the second chapter have to be considered as preliminary and a starting point for further development of a Graph Transformation System based machinery for describing the Tropos diagram generation and analysis process. The advantages we expect to be able to introduce with future works on the topic can be foreseen at least in two directions:

- the formal inspection of the analysis process in abstract and of other specific case studies may lead to careful definitions and comparison of different strategies of analysis, as preliminary exemplified in the present chapter.

- the precise definition of the Graph Transformation System and its formal analysis should allow us to prove that the System is algebraically close with respect to the notion of a valid Tropos diagram, as given is Section 2.2 and with Table 2.1. An immediate consequence would be that an implementation of a Tropos diagram editing tool, providing syntactic checking services, is possible simply by using Graph Transformation programming languages as, for example, PROGRESS [Sch91].

In the third chapter of the deliverable we have described our approach, based on GR, to support the visual modeling process in *Tropos*. We have given examples of the *Tropos*

rewriting rules which specify the syntax of the *Tropos* language and supports an analyst in building correct models. These rules have been implemented with AGG. The GR approach gives flexibility in adopting modeling language variants and can be adapted to allow for the definition of partially correct models. Moreover, derivation transformation seems to be a promising way to support the restructuring of a model in consistent ways. We are currently defining an appropriate notation for representing derivation history and completing the implementation and validation of the described techniques in AGG.

Notice that the work we have presented in the third chapter covers only the requirements phase. We intend to push forward the usage of graph transformations also in the later phases (in particular architectural design and detailed design) following the guidelines of the *Tropos* methodology [BGG+03]. A long term objective is that of integrating GR into a CASE tool aimed at supporting visual modeling within the *Tropos* methodology.

In the last chapter we have shown that in our extended Secure Tropos framework it is possible to automatically support the designer of secure and trusted systems also in the automatic selection of design alternatives. Our enhanced methodology allows to:

1. Capture through a graphical notation of the requirements for the organization and the system-to-be.
2. Verify the correctness and consistency of functional and security requirements by
   - completion of the model drawn by the designer with axioms (a process hidden to the designer),
   - checking the model for the satisfaction of formal properties corresponding to specific security or design patterns.
3. Automatically select alternative solutions for the fulfillment of functional and security requirements by
   - transformation of the model drawn by the designer into a planning problem (a process hidden to the designer),
   - automatic identification of an alternative satisfying the goals of the various actors by means of planner.

Finally we have shown that it is possible with the use of an off-the-shelf planner to generate possible designs for not trivial security requirements. Of course, we assume that the designer remains in the design loop, so the designs generated by the planner are seen as suggestions to be refined, amended and approved by the designer. In other words, the planner is a(nother) support tool intended to facilitate the design process.

Our future work includes extending the application of this idea to other phases of the design and towards progressively larger industrial case studies to see how far can we go without using specialized solvers.

Chapter 6

History of the Deliverable

Below we outline how the work described in the deliverable has evolved along the years of the project.

Change History

Version	Date	Status	Author (Unit)	Description
0.1	2003/Nov	Draft	ITC, UNITN	First version of the Tropos analysis process as graph transformation system.
0.2	2003/Nov	Draft	ITC, UNITN	First version of graph rewriting for agent oriented visual modeling.
0.3	2006/Nov	Draft	UNITN, ITC	First version of designing security requirements models through planning.
1.0	2006/Nov	Final	UNITN, ITC	Final revision of the deliverable.

Bibliography

[AEH+99] M. Andries, G. Engels, A. Habel, B. Hoffmann, H.-J. Kreowski, S. Kuske, D. Plump, A. Schurr, and G. Taentzer. Graph transformation for specification and programming. *Science of Computer Programming*, 1999.

[Axe84] Robert Axelrod. *The Evolution of Cooperation*. Basic Books, 1984.

[BCN92] Carlo Batini, Stefano Ceri, and Shamkant B. Navathe. *Conceptual database design: an Entity-relationship approach*. Benjamin-Cummings Publishing Co., Inc., Redwood City, CA, USA, 1992.

[BF97] A. Blum and M. L. Furst. Fast Planning Through Planning Graph Analysis. *Artif. Intell.*, 90(1-2):281–300, 1997.

[BGG+03] P. Bresciani, P. Giorgini, F. Giunchiglia, J. Mylopoulos, and A. Perini. *Tropos: An Agent-Oriented Software Development Methodology*. In Press, 2003.

[BGMS99] R. Bardohl, G.Taentzer, M. Minas, and A. Schurr. Application of Graph Transformation to Visual Languages. In *Handbook of Graph Grammars and Computing by Graph Transformation*, volume 2: Application, Languages and Tools. World Scientific, 1999.

[BMMZ06] V. Bryl, F. Massacci, J. Mylopoulos, and N. Zannone. Designing Security Requirements Models through Planning. Technical Report DIT-06-003, University of Trento, 2006.

[BPG+02] P. Bresciani, A. Perini, P. Giorgini, F. Giunchiglia, and J. Mylopoulos. Modelling early requirements in Tropos: a transformation based approach. In P. M.J.Wooldridge and G.Wei, editors, *Agent-Oriented Software Engineering II, LNCS 2222*, pages 151–168. Springer-Verlag,Montreal, Canada, Second InternationalWorkshop,AOSE2001 edition, May29th 2002.

[BS02] P. Bresciani and F. Sannicoloo. Applying Tropos Requirements Analysis for defining a Tropos tool. In *Agent-Oriented Information System. Proceedings of AOIS-2002: Fourth International Bi-Conference Workshop*, pages 135–138, Toronto, May 2002.

[Cal97] J. L. Caldwell. Moving proofs-as-programs into practice. In *ASE '97: Proc. of the 12th international conference on Automated software engineering*, page 10, Washington, DC, USA, 1997. IEEE Computer Society.

[CAS] CASE tools list odered by names: http://www.cs.queensu.ca/Software-Engineering/tools.html.

[CKM02] Jaelson Castro, Manuel Kolp, and John Mylopoulos. Towards Requirements-Driven Information Systems Engineering: The Tropos Project. *Inform. Sys.*, 27(6):365–389, 2002.

[CST03] M. Carman, L. Serafini, and P. Traverso. Web service composition as planning. In *Proc. of the 2003 Workshop on Planning for Web Services*, 2003.

[dGLL00] G. de Giacomo, Y. Lesperance, and H. J. Levesque. ConGolog, a concurrent programming language based on the situation calculus. *Artif. Intell.*, 121(1-2):109–169, 2000.

[Dru90] P. Drucker. *Managing the Non-Profit Organization: Principles and Practices*. HapperCollins Publishers, 1990.

[DvLF93] Anne Dardenne, Axel van Lamsweerde, and Stephen Fickas. Goal-directed requirements acquisition. *Science of Computer Programming*, 20(1-2):3–50, April 1993.

[ea06] P. Bresciani et al. KLASE MIUR-FIRB project RBAU01P5SS "Knowledge Level Automated Software Engineering", WP1.2. Technical report, University of Trento, 2006.

[EH04] S. Edelkamp and J. Hoffmann. Pddl2.2: The language for the classical part of the 4th international planning competition. Technical Report 195, University of Freiburg, 2004.

[Ell06] Thomas Ellman. Specification and synthesis of hybrid automata for physics-based animation. *Automated Software Engg.*, 13(3):395–418, 2006.

[ERT99] C. Ermel, M. Rudolf, and G. Taentzer. The AGG approach: Language and environment. In *Handbook of Graph Grammars and Computing by Graph Transformation*, volume 2: Application, Languages and Tools. World Scientific, 1999. See also AGG site: http://tfs.cs.tu-berlin.de/agg/.

[Gal01] M. J. Gallivan. Striking a balance between trust and control in a virtual organization: a content analysis of open source software case studies. *ISJ*, 11(2), 2001.

[GHJV95] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[GHK+98] M. Ghallab, A. Howe, C. Knoblock, D. McDermott, A. Ram, M. Veloso, D. Weld, and D. Wilkins. PDDL - The Planning Domain Definition Language. In *Proc. of AIPS'98*, 1998.

[gio01] Agent-oriented software development: A case study. In *Proceedings of the Thirteenth International Conference on Software Engineering - Knowledge Engineering (SEKE01)*, Buenos Aires, June 2001.

[GJKL01] G. Gans, M. Jarke, S. Kethers, and G. Lakemeyer. Modeling the Impact of Trust and Distrust in Agent Networks. In *Proc. of AOIS'01*, pages 45–58, 2001.

[GMMZ05a] P. Giorgini, F. Massacci, J. Mylopoulos, and N. Zannone. Modeling Security Requirements Through Ownership, Permission and Delegation. pages 167–176. IEEE Press, 2005.

[GMMZ05b] Paolo Giorgini, Fabio Massacci, John Mylopoulos, and Nicola Zannone. Modelling Social and Individual Trust in Requirements Engineering Methodologies. In *Proc. of iTrust'05*, volume 3477 of *LNCS*, pages 161–176. Springer-Verlag, 2005.

[gro] OMG group. Unified Modeling Language specification. Current version: 1.5. 2003. See also UML OMG site: http://www.omg.org/uml/.

[Gro01] Object Management Group. Model Driven Architecture (MDA). http://www.omg.org/docs/ormsc/01-07-01.pdf, July 2001.

[GT-] GT-VMT02 workshop site: http://www2.cs.fau.de/GTVMT02/.

[GZK03] M. Gogolla, P. Ziemann, and S. Kushke. Towards an Integrated Graph Based Semantics for UML. In *Electronic Notes in Theoretical Computer Science (eds. M. Bauderon and A. Corradini)*, volume 72. Elsevier, 2003.

[HLMN06] C. B. Haley, R. C. Laney, J. D. Moffett, and B. Nuseibeh. Using Trust Assumptions with Security Requirements. *Requirements Eng. J.*, 11:138–151, 2006.

[HM00] D. Hirsch and U. Montanari. Higher-Order Hyperedge Replacement Systems and their Transformations: Specifying Software Architecture Reconfigurations. In *Proceedings of the Joint APPLIGRAPH/GETGRATS Workshop on Graph Transformation Systems (GRATRA 2000)(eds. H. Ehrig and G. Taentzer)*, pages 215–223, 2000.

[HM02] D. Hirsch and U. Montanari. Two graph-based techniques for software architecture reconfiguration. In *Electronic Notes in Theoretical Computer Science (eds. M. Bauderon and A. Corradini)*, volume 51. Elsevier, 2002.

[KHN05] R. K.Woolthuis, B. Hillebrand, and B. Nooteboom. Trust, Contract and Relationship Development. *Organization Studies*, 26(6):813–840, 2005.

[KS92] H. Kautz and B. Selman. Planning as satisfiability. In *Proc. of ECAI'92*, pages 359–363. John Wiley Sons, Inc, 1992.

[Luh79] N. Luhmann. *Trust and Power*. Wisley, 1979.

[MCL+01] J. Mylopoulos, L. Chung, S. Liao, H. Wang, and E. Yu. Exploring alternatives during requirements analysis. *IEEE Software*, 18(1):92–96, 2001.

[MCN92] J. Mylopoulos, L. K. Chung, and B. A. Nixon. Representing and using non-functional requirements: A process-oriented approach. *IEEE Transactions on Software Engineering*, June 1992.

[MF99] J. McDermott and C. Fox. Using Abuse Case Models for Security Requirements Analysis. In *Proc. of ACSAC'99*, pages 55–66. IEEE Press, 1999.

[MPZ05] F. Massacci, M. Prest, and N. Zannone. Using a security requirements engineering methodology in practice: the compliance with the italian data protection legislation. *Computer Standards & Interfaces*, 27(5):445–455, 2005.

[MT] M. Matskin and E. Tyugu. Strategies of Structural Synthesis of Programs and Its Extensions. Comp. and Informatics, 20:1–25, 2001.

[MW80] Zohar Manna and Richard Waldinger. A deductive approach to program synthesis. *ACM Trans. Program. Lang. Syst.*, 2(1):90–121, 1980.

[MZ06] F. Massacci and N. Zannone. Detecting Conflicts between Functional and Security Requirements with Secure Tropos: John Rusnak and the Allied Irish Bank. Technical Report DIT-06-002, University of Trento, 2006.

[PBG+01] A. Perini, P. Bresciani, F. Giunchiglia, P. Giorgini, and J. Mylopoulos. A Knowledge Level Software Engineering Methodology for Agent Oriented Programming. In *Proceedings of the Fifth International Conference on Autonomous Agents*, Montreal, Canada, May 2001.

[Pee05] J. Peer. Web Service Composition as AI Planning - a Survey. Technical report, University of St. Gallen, 2005.

[PS03] A. Perini and A. Susi. Developing a Decision Support System for Integrated Production in Agriculture. *Environmental Modelling and Software Journal*, 2003. Submitted to.

[RB05] Steve Roach and Jeffrey Van Baalen. Automated procedure construction for deductive synthesis. In *ASE*, pages 12(4):393–414, 2005.

[RUP] Rational Unified Process product overview: http://www-306.ibm.com/software/awdtools/rup/.

[Sch91] Andy Schürr. Progress: A vhl-language based on graph grammars. In *Proceedings of the 4th International Workshop on Graph-Grammars and Their Application to Computer Science*, pages 641–659, London, UK, 1991. Springer-Verlag.

[Sch01] Andy Schürr. Adding Graph Transformation Concepts to UML's Constraint Language OCL. *Electr. Notes Theor. Comput. Sci.*, 44(4), 2001.

[Sim69] H. A. Simon. *The Science of the Artificial*. MIT Press, 1969.

[SO05] G. Sindre and A. L. Opdahl. Eliciting security requirements with misuse cases. *Requirements Eng. J.*, 10(1):34–44, 2005.

[TE00] A. Tsiolakis and H. Ehrig. Consistency Analysis of UML Class and Sequence Diagrams using Attributed Graph Grammars. In *Proc. of Joint APPLIGRAPH/GETGRATS Workshop on Graph Transformation Systems (eds. H. Ehrig and G. Taentzer)*, Berlin, March 2000.

[Tro] Tropos Project. site: http://www.troposproject.org.

[vBDJ03] Axel van Lamsweerde, Simon Brohez, Renaud De Landtsheer, and David Janssens. From System Goals to Intruder Anti-Goals: Attack Generation and Resolution for Security Requirements Engineering. In *Proc. of RHAS'03*, pages 49–56, 2003.

[Wel99] D. S. Weld. Recent Advances in AI Planning. *AI Magazine*, 20(2):93–123, 1999.

[YL01] E. S. K. Yu and L. Liu. Modelling Trust for System Design Using the i* Strategic Actors Framework. In *Proc. of the Workshop on Deception, Fraud, and Trust in Agent Societies*, pages 175–194. LNCS 2246, Springer-Verlag, 2001.

[Yu95] E. Yu. *Modeling Strategic Relationships for Process Reengineering.* PhD thesis, University of Toronto,Department of Computer Science, 1995.