The TROPOS Analysis Process as Graph Transformation System

Paolo Bresciani
ITC-irst
via Sommarive, 18
I-38050 Trento-Povo, Italy

Paolo Giorgini
DIT
University of Trento
via Sommarive, 14
I-38050 Trento-Povo, Italy

Abstract

Tropos is an agent-oriented methodology that covers software development from early requirements analysis to detailed design, allowing for a deeper understanding of the operational environment of the new software system. In earlier work we have characterized the process of early requirements analysis in terms of transformation applications. In this paper 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.

1 Introduction

Tropos [11, 6] 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-deep justification with respect to the organizational goals.

Therefore, Tropos considers much earlier phases than those supported in, for instance, OOP software engineering methodologies, such as those based on UML [7], where use case analysis is proposed as an early activity, followed by architectural design [10].

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 [4].

Early requirements analysis is concerned 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 nonfunctional 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 [3], we have 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, we focused on the definition of the transformations that can be applied for refining an initial Tropos model to a final one, working incrementally.

This is a very basic issue in defining a new methodology, as for instance proposed in [2] for Entity-Relationship schema design, in [5] for goal oriented requirements analysis, and in [8] for functional and non-functional requirements analysis.

In the present paper we focus on the redefinition of the transformation system for early requirements analysis, proposed in [3], 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.

As well, the definition of a formal and precise Graph Transformation System for describing the diagram transformation process in Tropos opens the possibility for an implementation of a Tropos diagram editing tool based on the use of a Graph Transformation programming language.

The paper is organized as follows. In Section 2, we recall the transformational based approach presented in [3]. In Section 3 we introduce the basic notions on Graph Transformation Systems, and show how the transformational based approach can be rephrased in terms of this formalism. Some execution algorithms with an example of execution are then introduced. Finally, Section 4 presents some conclusion.

2 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 [13]. 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 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 well spent (`taxes well spent`). One possible analysis of this situation produced the diagram of Figure 1, as illustrated in [3], 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[11, 6, 3]

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 environ-

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

[2]For most of the aspects analyzed in this paper, 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.

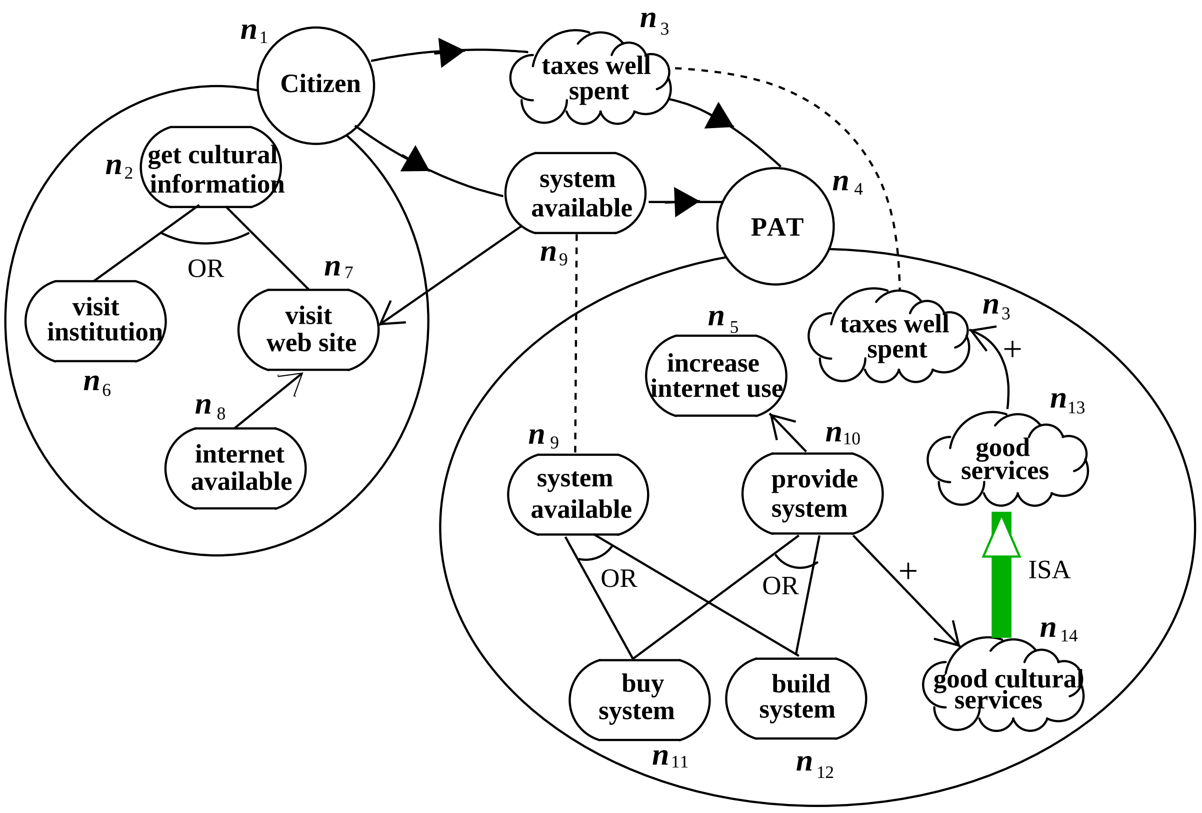

Figure 1: A Tropos actor diagram with rationale analyses for two actors (Citizen and PAT).

ment, and their introduction corresponds to a 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 [4], 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 [3], 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 al-

low 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 [8, 11]), 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 [3], but they are not dealt with in this paper.

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 [3] 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 1, that is derived from a case study presented more in detail in [3].

3 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 paper 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 [1].

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

Table 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 $\mathtt{T}$ is the set of possible types of nodes and edges, $\mathtt{T}$=\{*actor*, *goal*, *softgoal*, *resource*, *plan*, *and-decomposition*, *or-decomposition*, *precondition*, *generalization*, *assignment*, *outgoing-dependency*, *incoming-dependency*, *positive-contribution*, *negative-contribution*, *aggregation*\}[3], and $\mathtt{L}$ 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], $l_2(\mathcal{E}) = \{\varepsilon\}$[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 1, on the types of the nodes connected by an edge, must be observed.

3.1 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 [1]. 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.

[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.

[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.

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:

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

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:

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

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

We will also write $G \overset{P}{\Longrightarrow} H$, or $\overset{P}{\Longrightarrow} H$ 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 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 $G \overset{P}{\Longrightarrow} H$, 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.

[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 paper.

3.2 The Tropos diagram generation algorithm

Using the graph transformation system given in Table 1, the generic algorithm for driving the process of Tropos graph (or Tropos diagram) generation is given in Figure 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 [9].

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 in three categories. The 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.

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

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

Table 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.

```
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: The algorithm for the Tropos graph generation.

The new version of the algorithm is given in Figure 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 1.

Outer REPEAT:

First LOOP:

INTRODUCTION RULES:[10]

$\langle\{\},\{\}\rangle \overset{A\text{-}I}{\Rightarrow}_1 \langle\{n_1\},\{\}\rangle$

$\overset{G\text{-}I}{\Rightarrow}_2 \langle\{n_1,n_2\},\{n_2\to n_1\}\rangle$

[10]In the following derivations, for each generated graph only N is explicitly listed. The elements of E are not named; instead the two maps s and t are given in compact form by writing, e.g., $n_1 \to n_2$: in this case we mean that $s(e)=n_1$ and $t(e)=n_2$. The Function l can be easily read on Figure 1.

${}_3\overset{SG\text{-}I}{\Rightarrow} \langle\{n_1,n_2,n_3\},\{n_2\to n_1,n_3\to n_1\}\rangle$

$\overset{A\text{-}I}{\Rightarrow} \langle\{n_1,n_2,n_3,n_4\},\{n_2\to n_1,n_3\to n_1\}\rangle$

$\overset{G\text{-}I}{\Rightarrow} \langle\{n_1,n_2,n_3,n_4,n_5\},\{n_2\to n_1,n_3\to n_1,n_5\to n_4\}\rangle$

ANALYSIS RULES:[11]

$\overset{G\text{-}I^2}{\Rightarrow} \langle N\cup\{n_6,n_7\},E\cup\{n_6\to n_1,n_7\to n_1\}\rangle$

$\overset{G\text{-}DEC\text{-}OR}{\Rightarrow} \langle N,E\cup\{n_6\to n_2,n_7\to n_2\}\rangle$

$\overset{G\text{-}I^2}{\Rightarrow} \langle N\cup\{n_8,n_9\},E\rangle$

$\overset{G\text{-}P^2}{\Rightarrow} \langle N,E\cup\{n_8\to n_7,n_9\to n_7\}\rangle \overset{G\text{-}I}{\Rightarrow} \langle N\cup\{n_{10}\},E\rangle$

$\overset{G\text{-}P}{\Rightarrow} \langle N,E\cup\{n_{10}\to n_5\}\rangle \overset{G\text{-}I^2}{\Rightarrow} \langle N\cup\{n_{11},n_{12}\},E\rangle$

$\overset{G\text{-}DEC\text{-}OR^2}{\Rightarrow} \langle N,E\cup\{n_{11}\to n_{10},n_{12}\to n_{10}\}\rangle$

DELEGATION RULES:

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

$\overset{SG\text{-}DEL}{\Rightarrow} \langle N,\{n_2\to n_1,\mathbf{n_1}\to\mathbf{n_3},\mathbf{n_3}\to\mathbf{n_4},\mathbf{n_3}\to\mathbf{n_4},n_5\to n_4,n_6\to n_1,n_7\to n_1,n_6\to n_2,$
$n_7\to n_2,n_8\to n_1,n_1\to n_9,n_9\to n_2,n_9\to n_2,n_8\to n_7,n_9\to n_7,n_{10}\to n_4,n_{10}\to n_5,$

[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 N, and the set of edge so far included in the graph simply with E; N increases always monotonically, thus, only new nodes will be evidenced; 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.

```
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 3: Enhanced version of the algorithm for the Tropos Graph generation.

$n_{11} \rightarrow n_4, n_{12} \rightarrow n_4, n_{11} \rightarrow n_{10}, n_{12} \rightarrow n_{10}\}\rangle$[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:

$\overset{G\text{-}DEC\text{-}OR}{\Rightarrow} \langle N, E \cup \{n_{11} \rightarrow n_9, n_{12} \rightarrow n_9\}\rangle$

$\overset{SG\text{-}I}{\Rightarrow} \langle N \cup \{n_{13}\}, E \cup \{n_{13} \rightarrow n_4\}\rangle$

$\overset{SG\text{-}C\text{-}SG(+)}{\Rightarrow} \langle N, E \cup \{n_{13} \rightarrow n_3\}\rangle$

$\overset{SG\text{-}I}{\Rightarrow} \langle N \cup \{n_{14}\}, E \cup \{n_{14} \rightarrow n_4\}\rangle$

$\overset{SG\text{-}GEN}{\Rightarrow} \langle N, E \cup \{n_{14} \rightarrow n_{13}\}\rangle$

$\overset{SG\text{-}C\text{-}SG(+)}{\Rightarrow} \langle N, E \cup \{n_{10} \rightarrow n_{14}\}\rangle$

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.

For all these reasons, we propose, in Figure 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

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

```
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 4: The final version of the algorithm.

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

[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.

4 Conclusion

As presented in [3], 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 paper we have focused 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 an example of execution, and finally, we have discussed some preliminary observations on different control strategies.

The work and the considerations presented in this paper 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.

First, 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 paper.

Second, 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 3 and with Table 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 [12].

References

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

[2] C. Batini, S. Ceri, and S. Navathe. *Conceptual Database Design. An Entity-Relationship Approach*. Benjamin-Cummings Redwood City, Calif., 1992.

[3] 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, G. Weiß, editor, *Agent-Oriented Software Engineering II*, LNCS 2222, pages 151–168. Springer-Verlag, Montreal, Canada, Second International Workshop, AOSE2001 edition, May29th 2002.

[4] P. Bresciani and F. Sannicoloò. 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.

[5] A. Dardenne, A. van Lamsweerde, and S. Fickas. ”Goal” directed Requirements Acquisition. *Science of Computer Programming*, (20), 1993.

[6] P. Giorgini, A. Perini, J. Mylopoulos, F. Giunchiglia, and P. Bresciani. Agent-oriented software development: A case study. In *Proceedings of the Thirteenth International Conference on Software Engineering - Knowledge Engineering (SEKE01)*, Buenos Aires, jun 2001.

[7] I. Jacobson, G. Booch, and J. Rumbaugh. *Unified Software Development Process*. Addison-Wesley, 1999.

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

[9] 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.

[10] A. Perini, P. Bresciani, P. Giorgini, F. Giunchiglia, and J. Mylopoulo. Towards an Agent Oriented approach to Software Engineering. In *Proceedings of WOA 2001 – Dagli oggetti agli agenti: tendenze evolutive dei sistemi software*, Modena, sep 2001. Pitagora Editrice Bologna.

[11] 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.

[12] A. Schürr. PROGRESS: A VHL-language based on graph grammars. In H. Ehrih, H.-J. Kreowski, and G. Rozenberg, editors, *Proc. Graph Grammars and Their Application to Computer Science*, volume 532 of *LNCS*, pages 641–659. Springer, 1991.

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