Modeling early requirements in Tropos: a transformation based approach

Paolo Bresciani and Anna Perini
ITC-Irst
via Sommarive, 18
I-38050 Trento-Povo, Italy
telephone: +39 0461 314342
bresciani@irst.itc.it
perini@irst.itc.it

Paolo Giorgini and Fausto Giunchiglia
Department of Information and Communication Technology
University of Trento
I-38050 Trento-Povo, Italy
pgiorgini@cs.unitn.it
fausto@cs.unitn.it

John Mylopoulos
Department of Computer Science
University of Toronto
M5S 3H5, Toronto, Ontario, Canada
jm@toronto.edu

Keywords

Agent-oriented software engineering, agent-oriented requirements analysis, analysis methodologies.

ABSTRACT

We are developing an agent-oriented software development methodology, called Tropos, which integrates ideas from multi-agent system technologies and Requirements Engineering research. A distinguishing feature of Tropos is that it covers software development from early requirements analysis to detailed design, allowing for a deeper understanding of the operational environment of the new software system.

This paper proposes a characterization of the process of early requirements analysis, defined in terms of transformation applications. Different categories of transformations are presented and illustrated by means of a running example. These transformations are then mapped onto a set of primitive transformations. The paper concludes with observations on the form and the role of the proposed transformations.

1. INTRODUCTION

We are working on an agent-oriented software development methodology called Tropos[1], that integrates ideas from multi-agent system technologies, mostly to define the implementation phase for the software development process [2]; and ideas from Requirements Engineering, where actors and goals have been used heavily for early requirements analysis [5, 10]. In a recent state of the art survey on agent-oriented software engineering [4] several principled but informal methodologies have been considered together with approaches based on the use of formal methods for engineering multi-agent systems. Tropos shares with these approaches several methodological principles. In addition, it also features both diagrammatic and formal specification techniques, allowing for application of formal analysis techniques, such as model checking [6] and formal verification [11].

More concretely, Tropos is based on two key ideas. First, the notion of agent and all the related mentalistic notions (for instance: beliefs, goals, actions and plans) are used in all phases of software development, from the early analysis down to the actual implementation. Second, Tropos covers also the very early phases of requirements analysis, thus allowing for a deeper understanding of both the environment where the software must operate and the kind of interactions that should occur between software and human agents. In particular, Tropos rests on five main phases for agent based systems development: the early requirements analysis, the late requirements analysis, the architectural design, the detailed design, and the implementation. A complete example of the application of the Tropos methodology to a realistic case-study is presented in [8].

Following current approaches in software development, Tropos rests on the use of a conceptual model of the system-to-be. The conceptual model is specified by a modeling language resting on Eric Yu's $i*$ paradigm [12] which offers actors, goals, and actor dependencies as primitive concepts for modeling an application during the requirements analysis. We are currently working on the specification of the Tropos modeling language. A preliminary definition can be found in [9] where a meta-model of the language is given. At present, the graphical notation is largely borrowed from $i*$. Using this notation, different ways for visualizing the structural properties of the models can be introduced, as e.g., the actor diagram, that mainly shows actor dependencies, and the rational diagram, that provides a rationale of actor dependencies, through, for instance, a goal analysis conducted from the point of view of a specific actor [8].

In the present paper we focus on the definition of the transformations that can be applied for refining an initial Tropos model to a final one, working incrementally. This is, in general, a very basic issue, when defining a new method-

[1]Tropos (from the Greek "$\tau\rho o\pi\eta$", "tropé", which means "easily changeable", also "easily adaptable").

ology, as reported in works on Entity-Relationship schema design [1], goal oriented requirements analysis [5], and functional and non-functional requirements analysis [7]. Here, we describe the model transformation process that corresponds to the first phase in Tropos methodology —the early requirements analysis. This phase concerns 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 rational for the specification of the functional and non-functional requirements of the system-to-be, that will be defined during the subsequent late requirement analysis.

In section 2, a running example is introduced, for being used in successive sections. Section 3 discusses how the conceptual model definition and refinement process can be described in term of transformation applications. Different categories of transformations are introduced in sections 4, 5, 6. Finally, in section 7, we provide a minimal set of *primitive* transformations which can be used as a base for describing compound transformations, as those used in the running example. In section 8, we discuss some open points and future research directions.

2. A RUNNING EXAMPLE

The example considered refers to a web-based broker of cultural information and services (the *eCulture System* [8]), developed for the government of Trentino (Provincia Autonoma di Trento, or PAT). The system is aimed at presenting integrated information obtained from museums, exhibitions, and other cultural organizations and events, to the users, that are both citizens and tourists looking for things to do.

During the development process, the requirements engineer begins the analysis identifying the main stakeholders, and their desires and intentions. In Tropos, stakeholders are modeled as actors who may depend on other ones for goals to be achieved, tasks to be performed, and resources to be provided. Desires and intentions are modeled as goals and softgoals.[2].

The list of relevant actors for the eCulture project includes, among others, the following stakeholders:

- Provincia Autonoma di Trento (PAT) is the government agency funding the project; their objectives include improving public information services, increasing tourism through new information services, also encouraging Internet use within the province;

- Museums are major cultural information providers for their respective collections;

- Tourists want to access cultural information before or during their visit to Trentino;

- (Trentino's) citizens want easily accessible information, of any sort, and (of course) good administration of public resources.

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

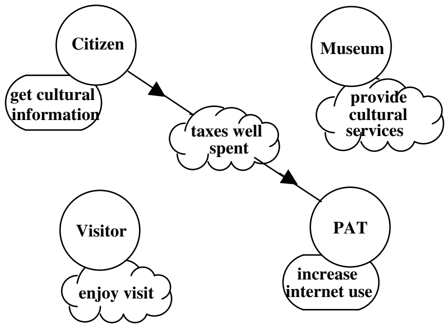


Figure 1: The stakeholders of the eCultural project

Figure 1 shows the initial early requirements model in form of actor diagram. An actor diagram is a graph, where each node represents an *actor*, and each link between two actors indicates that one actor depends on the other for something in order that the former may attain some goal. We call the depending actor the *depender* and the actor who is depended upon the *dependee*. The object around which the dependency centers is called the *dependum*. By depending on another actor for a dependum, an actor is able to achieve goals that it is otherwise unable to achieve on its own, or not as easily, or not as well. At the same time, the depender becomes vulnerable. If the dependee fails to deliver the dependum, the depender would be adversely affected in its ability to achieve its goals. So, for instance, in Figure 1 `Citizen` is associated with a single relevant goal: `get cultural information`, while `Visitor` has an associated softgoal `enjoy visit`. Softgoals are distinguished from goals because they don't have a formal definition, and are amenable to a different (more qualitative) kind of analysis [3]. Along similar lines, `PAT` wants to `increase internet use` while `Museum` wants to `provide cultural services`. Finally, the diagram includes one softgoal dependency where `Citizen` depends on `PAT` to fulfill the `taxes well spent` softgoal.

3. CONCEPTUAL MODEL TRANSFORMATIONS

As introduced in section 2, during the early requirement analysis, the engineer models the set of desires and intentions, informally expressed by the stakeholders, in terms of goals and dependencies between actors. This is an iterative process, at each step of which details are incrementally added and rearranged starting from a rough version of the model, containing only few actors, goals, softgoals, and dependencies, as, for example, shown in Figure 1. 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 deeper and more precise understanding of the requirements. Of course, to accomplish this task, the engineer might need to acquire new information or interact with the stakeholders. In the present paper, these aspects are not dealt with, and we assume that the relevant knowledge is somehow available to the engineer.

The process of conceptual modeling, that is the core part of the early requirement analysis phase, can be described in terms of simple transformations of subsequent versions of the model. Each of these transformations allows the progressive introduction of more structure and details in the model. In other words, by iterating the application of transformations, the engineer can move from a very sketchy first version to the final complete model, going through subsequent, more and more precise and detailed, versions. At each step of the process several transformations are, in general, applicable. It is up to the engineer deciding which transformation to apply: different choices result in different strategies for the modeling activity. The analysis of possible different strategies is, indeed, a topic that would deserve its own space for discussion. In the present paper, due to lack of room, we do not tackle this argument.

In the following sections, the most relevant transformations are introduced, under the form of generic rules transforming patterns in the model into more complex configurations. The transformations are illustrated through their application to the already mentioned case study. Moreover, they are grouped accordingly to the fact that they are used for actor, goal, or softgoal analysis. Another orthogonal classification is represented by the *role* that the transformation application plays in the actual process: we distinguish between *top down* (TD) and *bottom up* (BU) applications. Applying transformations in a TD 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 w.r.t. other conceptual elements. Vice versa, BU 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. This notion of top down and bottom up partially differs from that given by other authors in different modeling contexts (see [1]). In fact, they mainly mean top down as "adding more elements" in the schema vs. "linking existing elements" (bottom up), with the only exception for the only generalization relationship (ISA between entities). In our context, instead, other kinds of *abstractions* are possible, e.g., of the kind of *aggregation* and *composition*, and, in some sense, also *softgoal contributions*, allowing for a wider range of *genuine* ways of applying transformations in top down or bottom up.

Indeed, strictly speaking, each transformation application should be considered as a symmetric operation that introduces a new conceptual relationship between two (sets of) elements, and no distinction between TD and BU should be meaningful. Nevertheless, it must be noted that, during the modeling process, the engineer focuses, in turn, on different (sets of) conceptual elements of the model. If we consider whether the (set of) element(s) currently under analysis (from here, simply "u.a.") is the most generic or the most specific among those involved in the transformation, a *direction* —TD or BU, respectively— emerges. Focusing in turn on the different elements is an essential aspect of the methodology. For this reason, it is important distinguishing between the two directions as relevant features for the strategy of the process. This distinction is explicit in Tables 1, 2, 3, as well as in the running example that illustrates the transformations in the following sections.

Finally, a deeper analysis of the transformations allows us to note that, although they correspond in a natural way to the steps made by the engineer in order to build the model, they cannot be considered as atomic. A smaller set of simpler, although less operable, transformations may be introduced: we call them *primitive transformations*. At this level of granularity, the distinction between TD and BU is, of course, meaningless.

4. GOAL TRANSFORMATIONS

Goal transformations (see Table 1) allow us to perform goal analysis by introducing relationships between goals, or actors and goals. Of course, relationships between conceptual elements already present in the model can be introduced, but the process can also lead to the introduction of previously unconsidered conceptual elements. In this case, the diagram is enriched not only with new structure (in terms of links between items), but also with new items. In a Tropos models, goals must always be associated to at least one actor.[3] Thus, it necessary to define a *default* actor dependency for the introduced goals. Our assumption is that the new goals initially belongs and depend on the same dependee of the goal(s) u.a. This implies that, if we have more goals u.a. (as in some BU cases), they must share the same dependee.

4.1 Goal Decomposition

Goal decomposition transformations allow for the decomposition of a goal in and/or subgoals s.t. the subgoal achievements "automatically" implies the goal achievement. In other terms, the achievement of one (for the *OR-decomposition*) or all (for the *AND-decomposition*) subgoals, corresponds to the achievement of the top (decomposed) goal.

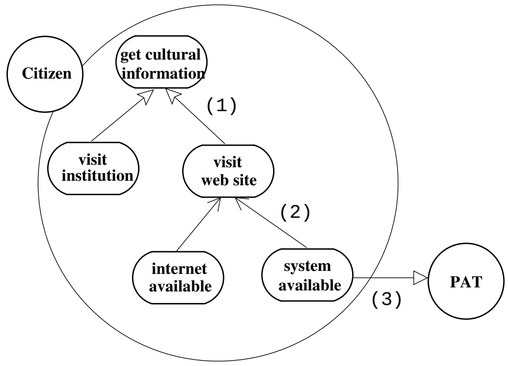


Figure 2: Citizen's goals analysis

For example, consider Figure 2, where, starting from the situation depicted in Figure 1, some goal analysis on `Citizen`'s goals is carried on. An instance of goal decomposition is here applied to the goal `get cultural info` (step **(1)**) in order to OR-decompose it into the two possible subgoals `visit institution` and `visit web site`.

In parallel with both OR-decomposition and AND-decomposition transformations, that are classified as TD, also the corresponding BU transformations can be given, namely *OR-composition* and *AND-composition* (see Table 1). An example is given in Figure 3, subsection 5.1.

[3] That is, the depender of a goal must alway be defined.

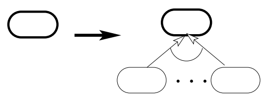
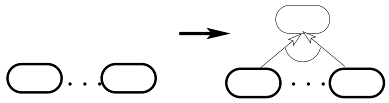
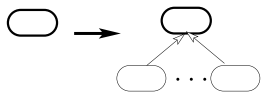
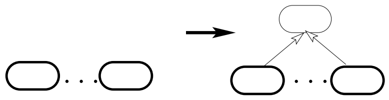
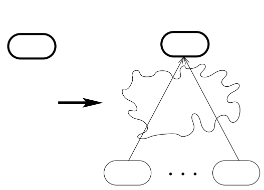
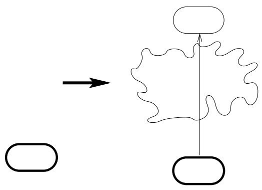
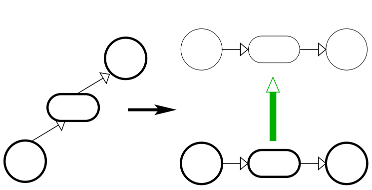
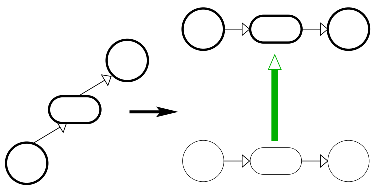
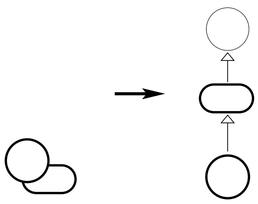
Goal AND-Decomposition (TD)	Goal AND-Composition (BU)
	
Goal OR-Decomposition (TD)	Goal OR-Composition (BU)
	
Precondition Goals (TD)	Precondition Goals (BU)
	
Goal Generalization	Goal Specialization
	
Goal Delegation	
	

Table 1: Goal Transformations

4.2 Precondition Goals

Precondition goal transformations substantially differ from goal decomposition transformations. They 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. Typically, goal analysis obtained by the application of precondition goals transformations has to be somehow completed, with more elements provided by further goals analysis and/or task analysis, that are part of later Tropos phases like the late requirement analysis[4].

Proceeding with our example, we can think of providing some preconditions for goal `visit web site`, introducing the precondition dependencies marked with **(2)** in Figure 2. This step corresponds to recognizing that the fulfillment of goals `internet available` and `system available` are necessary, although non sufficient, preconditions for the fulfillment of the goal `visit web site`.

4.3 Goal Delegation

Goal delegation transformations are aimed at allowing to express the assignment or a change of responsibility in goal fulfillment. In Tropos methodology, when initially defined, each goal is either expressed as a dependum between two actors or it is associated with an actor, who *desires* its fulfillment. As already mentioned, in the first case the dependee is assumed to be committed for the fulfillment of the goal. In the second case, instead, further analysis is required to the engineer in order to define which actor (possibly the same one) is committed to the fulfillment of the goal on behalf of the first. After this level of detail is reached, the goal is considered as an *intention* of the committed actor. This step in the process is called goal delegation transformation (including the *goal self-delegation transformation* as a special case). The goal delegation transformation can be applied to a goal and the actor it is initially assigned to (that is, the actor that desires the goal), as shown in Table 1, but also to a goal and the delegated actor, in order to obtain a *cascade* of delegations: in this case the *intention* is transferred (delegated) to another actor.[5]

Using our running example, we can see an example of goal delegation in Figure 2, where the responsibility for the goal `system available` is delegated to `PAT` (transformation number **(3)** applied to `Citizen` and its goal `system available`).

Summarizing, after the application of goal decomposition, precondition goal and goal delegation transformations to the model containing only `Citizen` and `get cultural info`, that is, namely, after steps **(1)**, **(2)** and **(3)**, our model is evolved as depicted in Figure 2.[6]

4.4 Goal Generalization

In some situations it can be useful to apply *goal specialization* or *goal generalization* transformations. In these cases, goal depender and dependee are, by default, inherited both for the TD and the BU transformations. When, instead, they are already defined, the ISA hierarchy introduced for the two goals must also hold between their respective dependers and dependees.

[4]Late requirement analysis and the role of task analysis is briefly introduced in [8].

[5]It can be noted that, also in this case, the transformation could require the introduction of new elements in the model, as well as just introducing the dependency between existing elements.

[6]The numbers in bold and parenthesis "**(1)**", "**(2)**" and "**(3)**" are not part of the model, but are here used only for describing the process.

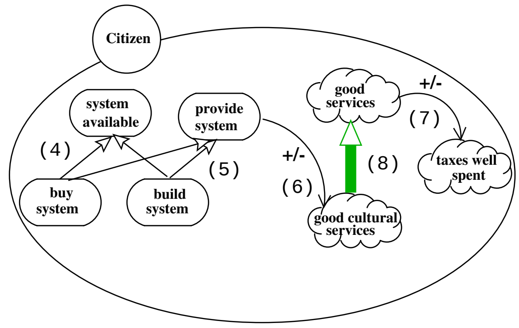


Figure 3: PAT's goals analysis

5. SOFTGOAL TRANSFORMATIONS

For softgoal analysis, transformations similar to some of those listed for goal analysis are available. Also the default actor assignment follows the rules already stated for goal analysis.

Moreover, for softgoals, during early requirement analysis, also a contribution analysis is performed (see [7, 8]). It is therefore necessary to foresee some specific transformations.

The transformations for softgoal analysis are listed in Table 2. Below, only few remarks and examples are given.

5.1 Softgoal Contribution

Contribution transformations allow us to specify whether a goal or soft-goal contributes to some other softgoal (BU) or, starting from the other side, whether there is some goal or soft-goal that contributes (TD) positively or negatively to the softgoal satisficement.

Two examples of application of contribution transformations are shown in Figure 3, namely, steps **(6)** (BU) and **(7)** (TD). It is also worth noticing that new instances of goal decomposition have been applied, once in a TD way (step **(4)**) and once in BU direction (step **(5)**).

5.2 Softgoal Decomposition

Softgoal decomposition transformations allow us to perform softgoal – sub-softgoals (or super-softgoals) analysis. Also in this case we distinguish between *softgoal decomposition* (TD) and *softgoal composition* (BU) (see Table 2).

While for goal analysis the decomposition gives a set of necessary subgoals (AND decomposition), or alternatives subgoals (OR decomposition), the fulfillment of which has to be considered as necessary and sufficient condition for the fulfillment of the higher level goal, for softgoal analysis the satisficement of the decomposition can never be considered as a sufficient condition, but only as a (positive) contribution.

5.3 Softgoal Delegation

We list *softgoal delegation transformations* here for completeness, but nothing different from what already written for goal delegation transformations has to be added.

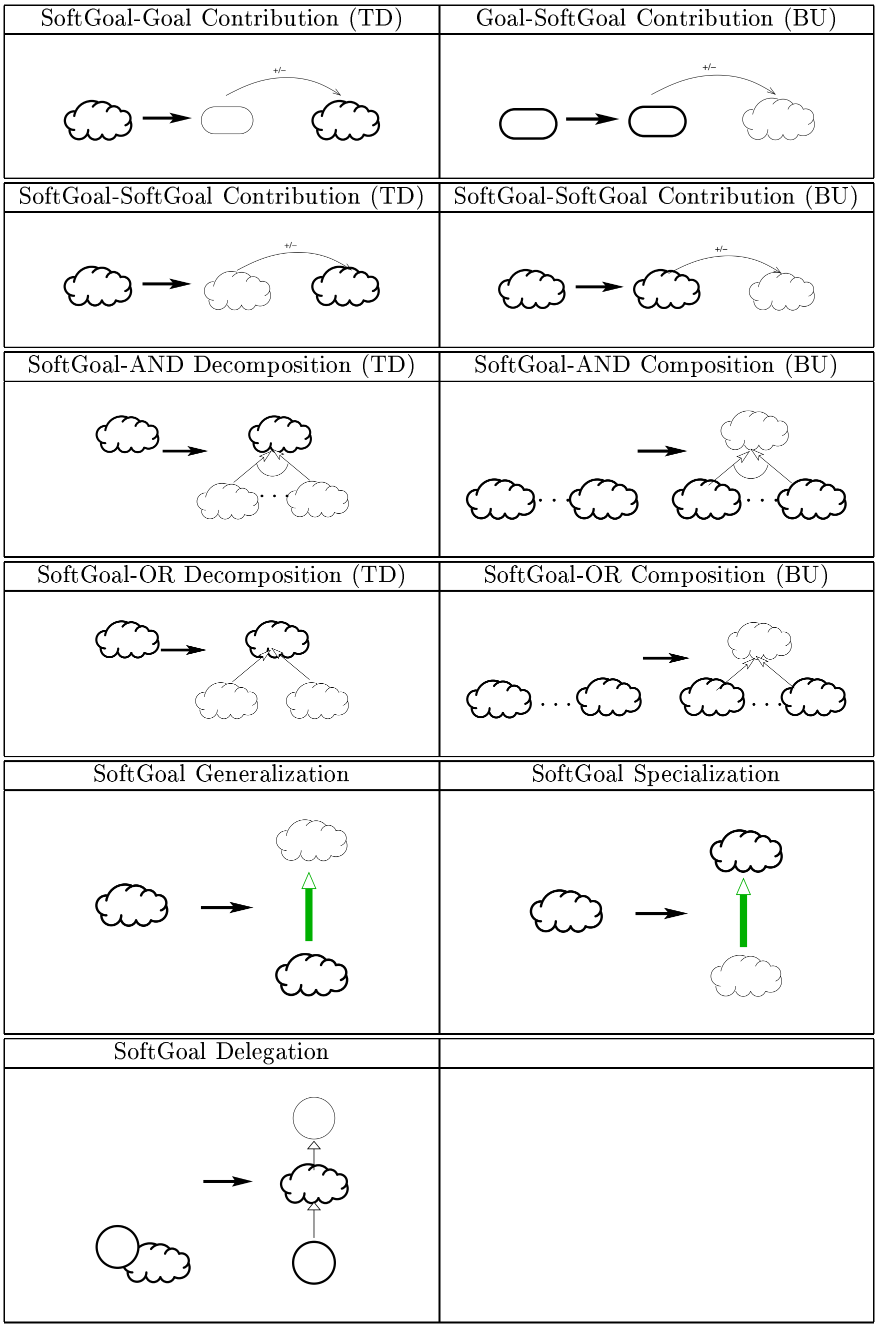


Table 2: SoftGoal Transformations

5.4 Softgoal Generalization

The same observations made for goal generalization transformations could be repeated here. Moreover, an example of application of a *softgoal generalization transformation* can be shown. Consider Figure 3: step **(8)** introduces a softgoal generalization between the softgoal `good cultural services` and the the softgoal `good services`. In this way, using the structured analysis introduced so forth —including, in particular, step **(7)**— it is evidenced that, fulfilling a possible set of goals required for the fulfillment of the goal `get cultural info`, originally under analysis, provides also a way for positively contributing to the satisfaction of the softgoal `tax well spent`.

6. ACTOR TRANSFORMATIONS

Actor transformations are only actor aggregation and actor generalization (see Table 3).

6.1 Actor Aggregation

Aggregating actors means recognizing that different actors are part of an organization or a system. This way of grouping can be helpful in order to more precisely delegate goals and softgoal responsibilities that are initially assigned to the aggregate. Thus, goal/softgoal delegation from an aggregate to its inside components or among components of the same aggregate should be preferred to other kinds of delegations. Of course, this preference can be recognized only once the components have been defined as such. For example, as further step in our running example, `PAT` could be *decomposed* in departments, and responsibility for each subgoal delegated to the appropriate department.

6.2 Actor Generalization

Actor generalization transformations can be used to introduce taxonomic structure among actor types. The engineer must be advised that the ISA relationship is meaningful as far as actor types are concerned as more generic element. If the actor u.a. is instead an instance, only a BU generalization transformation can be applied, or aggregation transformations have probably to be considered.

As for goal generalization transformations, it is worth noticing that possible existing goal dependencies with the involved actors must involve goals that respect the actor ISA hierarchy.

In our example, although not reported in any figure, consider the case in which the engineer wants to classify `PAT` as a `Government institution`. In this case, all the goals (or specialized versions of them) possibly deriving from previously made analysis on `Government institution`, can automatically be inherited by `PAT`.

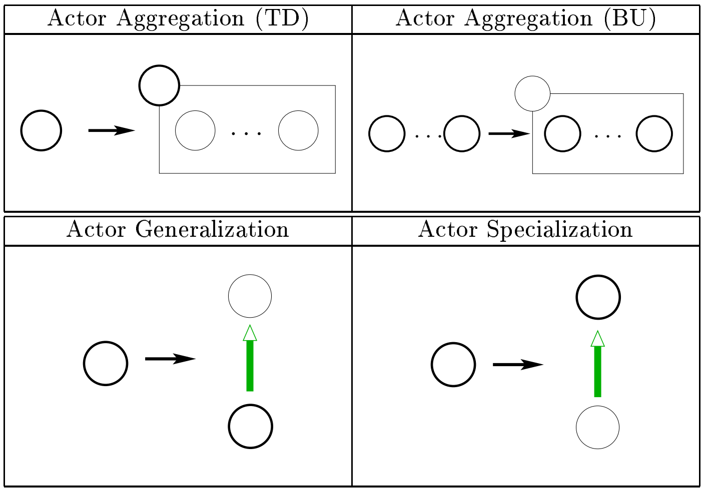


Table 3: Actor Transformations

7. PRIMITIVE TRANSFORMATIONS

As mentioned in section 3, the set of transformations presented in the previous sections is largely redundant. The reason why they have been presented here is that, from a practical point of view, they reflect the approaches the engineer may locally adopt for analyzing, in turn, the different conceptual elements of the model. By applying TD transformations, the engineer breaks the problem into more manageable sub-elements: the analysis has to be carried on until a sufficient level of details in reached for each necessary goal (originally present or later introduced) and the most specific components are likely to be easily operationalized, in terms of tasks analysis, by means of the successive *late requirement* phase.[7] For softgoals, a *reasonable* level of satisficeability has to be reached. Of course, TD steps can be interleaved with BU transformation applications. They are used, e.g., for structuring scattered models and for taking advantage of analysis made for parts of the model, w.r.t. different, but interacting, elements of the model.

Here, it can be useful to briefly show that a smaller set of transformations can be provided, that is enough, although in a less compact way, to perform any kind of analysis provided by the more structured transformations introduced before. Moreover, each of these transformations must be such that it cannot be expressed as the composition of other transformations. For this reason we call these transformations *primitive*.

The set contains a primitive transformation for each kind of relationship that can be introduced between the conceptual elements. In addition, *initial* transformations for introducing the different conceptual elements in the model are needed. Of course, as already noted, the distinction between TD and BU is not applicable for primitive transformations.

The set of primitive transformations is give in Table 4. It must be remembered that, accordingly with Tropos methodology, a goal or a softgoal is always present in the model associated with an actor (or two, when delegated). This justify for the form of the **G-I** and **SG-I** primitive transformations. Thus, the only primitive transformation available for starting populating an empty model is the **A-I** transformation.

Using the listed primitives, assuming as syntactic convention that they can be applied to conceptual elements, and, for the introduction primitives, to names of the elements to be introduced, the example of model development process described in the previous sections can be rewritten as shown in Figure 4. It must be noted that almost each step corresponds to the composition of some primitive applications.

[7] Of course, also inadequate evaluations can be expected to be done in this transition step, but it must also be said that the two phases are permeable, and backward jumps are always possible.

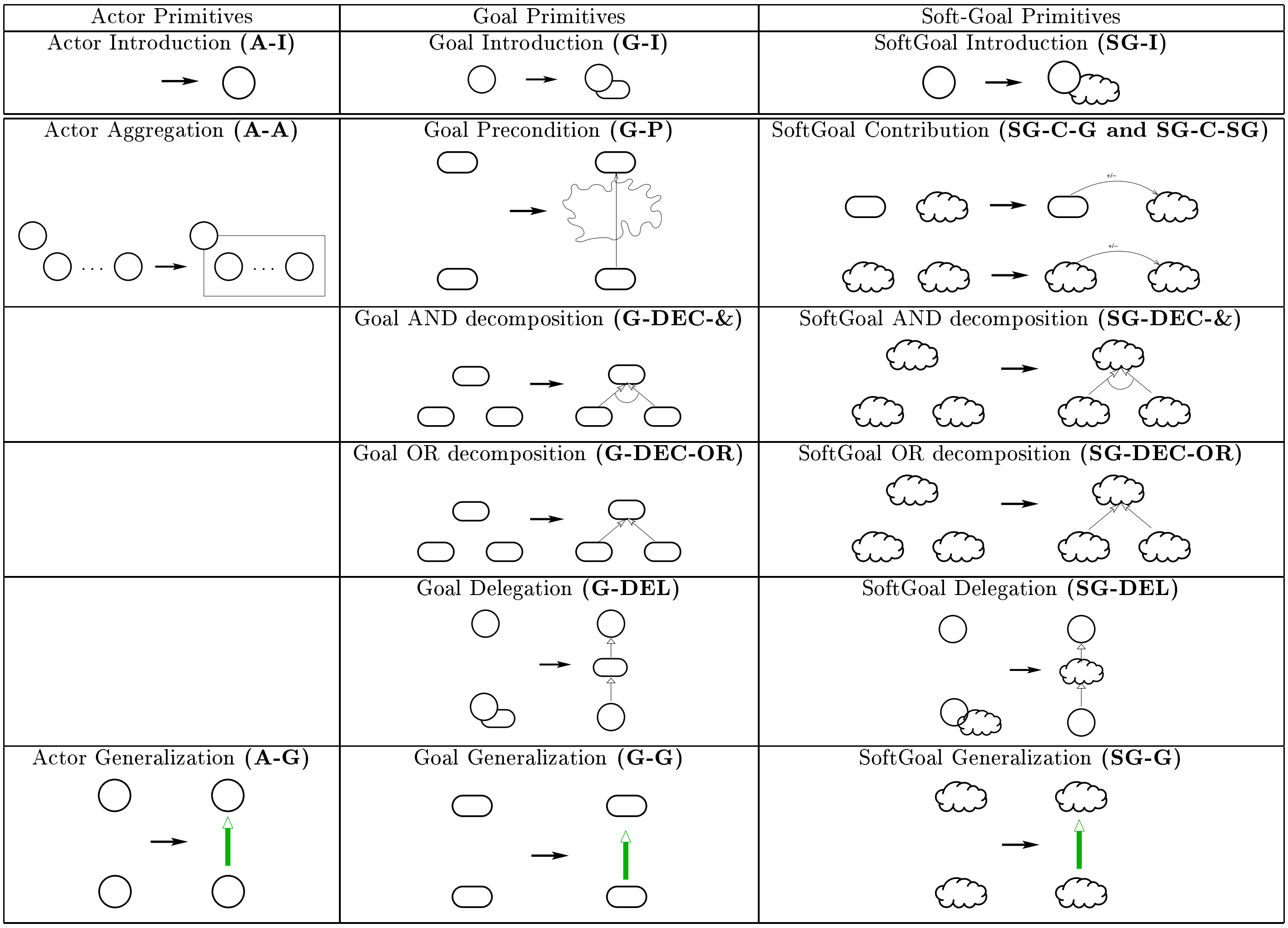


Table 4: SoftGoal Transformations

1. **G-DEC-OR**(`Get-Cultural-info`,**G-I**(`Citizen`,"visit institution"),**G-I**(`Citizen`,"visit web site"))
2. **G-P**(`visit web site`,**G-I**(`Citizen`,"internet available"),**G-I**(`Citizen`,"system available"))
3. **G-DEL**(`system available`,`PAT`)
4. **G-P**(`system available`,**G-I**(`PAT`,"buy system"),**G-I**(`PAT`,"build system"))
5. **G-DEC-&**(**G-I**(`PAT`,"provide system"),`buy system`,`build system`)
6. **SG-C-G**(**SG-I**(`PAT`,"good cultural services"),`provide system`,+)
7. **SG-C-SG**(`tax well spent`,**SG-I**(`PAT`,"good services"),+)
8. **SG-G**(`good services`,**G-I**(`good cultural services`))

Figure 4: The example described with primitive transformations

8. CONCLUSION

In this paper we have presented one aspect of our research aimed at defining a comprehensive methodology, called Tropos, for agent oriented software engineering. In particular, we dealt with the early requirement analysis phase, that, in Tropos, is used for defining the environment of the system-to-be. The activity of the requirement engineer has been described as an iterative process based on successive transformations that allow her to progressively define and refine the model of the social environment of the system-to-be. A set of transformations has been described, and illustrated through a running example. Finally, this set of practically usable transformations has shown to be reduceable to a more limited, but less usable, set of primitive transformations.

From our analysis, two main issues arises. First, the proposed transformation can be considered as the building block for guiding the engineer's activity. Moreover, the way they are used must be analyzed not only locally, but also w.r.t. the strategic role they play in the process. The distinction between TD and BU applications is only a first attempt in this direction. More in general, different uses of the transformations correspond to different developments of the activity of the engineer. If we consider the activity of the engineer as a problem solving activity[8], the problem solution can be seen as the result of the execution of a plan that the engineer elaborate in response to a precise goal. Of course, many details may be not clearly stated with the problem —engineering activity often include a considerable informal and human contribution— but it is essential the fact that the solution can be found after the application of a sequence of elementary steps. Using different sequences can lead both to different solutions and to the same solution, but with different performances in finding it. Tackling the analysis of the process as a planning activity is one of the next development of our research; in the present paper we only defined the single actions that can be used in a plan.

A second aspect that emerged from our analysis is that there is a difference between transformations that can be proposed as practically usable by the engineer, and primitive transformations, that, from a more formal point of view can be considered as the essential basis for defining any process. How primitive transformations can be combined in order to obtain *usable* transformation, and the role they may have in the process seen as a planning activity, is also a topic that has to be further investigated, possible with more case studies.

Of course, another natural development of the present work is its extension to other phases of the Tropos methodology, in particular to the late requirement analysis, that includes the development more details on the system-to-be, as its decomposition into components and their task analysis.

9. REFERENCES

[1] C. Batini, S. Ceri, and S.B.Navathe. *Conceptual Database design. An entity-relationship approach.* Benjamin-Cummings Redwood City, Calif., 1992.

[2] P. Busetta, R. Rönnquist, A. Hodgson, and A. Lucas. Jack intelligent agents - components for intelligent agents in java. AOS TR9901, Jan. 1999. http://www.jackagents.com/pdf/tr9901.pdf.

[3] L. K. Chung, B. A. Nixon, E. Yu, and J. Mylopoulos. *Non-Functional Requirements in Software Engineering.* Kluwer Publishing, 2000.

[4] P. Ciancarini and M. Wooldridge. *Agent-Oriented Software Engineering.* Springer-Verlag Lecture Notes in Computer scienceVolume 1957, 2001.

[5] A. Dardenne, A. van Lamsweerde, and S. Fickas. "goal" directed requirements acquisition. *Science of Computer Programming*, (20), 1993.

[6] A. Fuxman, M. Pistore, J. Mylopoulos, and P. Traverso. Model checking early requirements specifications in tropos. Technical report, irst, 2001. http://sra.itc.it/people/pistore/papers/FT.ps.gz.

[7] J. Mylopoulos, L. Chung, S. S. Y. Liao, H. Wang, and E. Yu. Exploring alternatives during requirements analysis. *IEEE Software*, 2001.

[8] 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*, Montral CA, May 2001. To appear.

[9] A. Perini, F. Sannicolo, and F. Giunchiglia. The Tropos modelling language specification. version 1.0. Technical report, irst, Feb. 2001. http://www.science.unitn.it/~pgiorgio/aose.

[10] A. van Lamsweerde. Requirements engineering in the year 00: A research perspective. In *Proceedings 22nd International Conference on Software Engineering, Invited Paper*, ACM Press, June 2000.

[11] X. Wang and Y. Lesperance. Agent-oriented requirements engineering using ConGolog and i*. 2001. Submitted to AOIS-2001, Bi-Conference Workshop at Agents 2001 and CAiSE'01.

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

[8] An informal way in which we can define the engineer's problem is: "Define, using appropriate knowledge (the acquisition of which is left out of the problem setting), the environment for the system-to-be, with a sufficient level of details".