ITC irst

CENTRO PER LA RICERCA
SCIENTIFICA E TECNOLOGICA

38050 Povo (Trento), Italy
Tel.: +39 0461 314312
Fax: +39 0461 302040
e−mail: prdoc@itc.it − url: http://www.itc.it

APPLYING TROPOS TO REQUIREMENT ANALYSIS FOR A TROPOS TOOL

Bresciani P., Sannicolo' F.

April 2002

Technical Report # 0204−01

Applying Tropos to requirement analysis for a Tropos tool

Paolo Bresciani and Fabrizio Sannicolò
ITC-Irst
Via Sommarive, 18
I-38050 Trento-Povo, Italy
{bresciani,sannico}@irst.itc.it

Abstract

Tropos, a novel agent-oriented software engineering methodology, is characterized by three key points: (i) it pays much attention to the activities that precede the specification of the prescriptive requirements, such as understanding how the intended system would meet the organizational goals; (ii) it uses the same mentalistic notions of actors, goals, plans, and actors' intentional dependencies, along all the phases of requirement analysis and system design; (iii) it foresees a process of requirement and system modeling that is incremental and iterative, based on a set of progressive transformational steps.

As a result, Tropos offers a much more homogeneous approach for the different development phases than most other approaches, also providing for a powerful support for the intentional analysis of actor' requirements and their correspondence with the adopted solutions and for a consistent use of traceability.

The present paper will take into account the application of the Tropos methodology to a *self-motivating* case study: the definition of a support tool for the Tropos methodology itself. Unlike in previous papers, here the attention will be concentrated on the early requirements and on how to manage the transition from this to the late requirement analysis. In other terms, the focus will mainly be on point (i), among those listed above, and on the high level characteristics of the iterative transformational process mentioned at point (iii), also as applied to the transition from early requirements to late requirements.

1 Introduction

Tropos is a novel agent-oriented software engineering methodology characterized by three key aspects.

First, it pays attention to the activities that precede the specification of the prescriptive requirements, such as understanding how the intended system would meet the organizational goals. This kind of approach was first proposed in the requirements engineering literature (see for instance [6, 20]). In particular, Tropos adopts ideas from the Eric Yu's model for requirements engineering, called i^*, which offers actors, goals, and actor dependencies as primitive concepts [20].[1]

A second key feature of Tropos is dealing with all the phases of system requirement analysis (the analysis, or requirement engineering, phases) and with all the phases of system design and implementation (the design, or software engineering, phases) in a uniform and homogeneous way, based on common mentalistic notions as those of actors, goals, plans, and intentional dependencies.

[1] i^* has been applied in various application areas, including requirements engineering [19] and software modeling processes [22].

Third, the Tropos methodology rests on the idea of building a model of the system-to-be that is incrementally refined and extended from a conceptual level to executable artifacts. This process adopts a transformational approach: a set of incremental transformation steps must be performed by the engineer to progressively detail the higher level notions introduced in the earlier phases [2]. Because, contrarily to what happens in most other approaches, like, for example, the UML based methodologies, in Tropos there is no sudden change of graphical notation from one step to the next, the refinement process is performed in a more uniform way [13].

The main goal of the Tropos methodology is to provide a more systematic reengineering of processes. One of the main advantages is that, by doing the intentional analysis required by Tropos, one can also capture not only the *what* or the *how*, but also the *why* a piece of software is developed. This, in turn, allows for a more refined analysis of the system dependencies and, in particular, for a much better and uniform treatment not only of the system functional requirements, but also of the non-functional requirements (the latter being usually very hard to deal with).

Tropos, although not only addressed to Agent Oriented Programming (AOP) [13], is best fit to AOP. In fact, the application of Tropos to AOP, and in particular the decision to use mentalistic notions in all the phases of analysis, has important consequences. When writing agent oriented specifications and programs one uses the same notions and abstractions used to describe the behavior of the human agents, and the processes involving them. Thus, the conceptual gap from *what* the system must do and *why*, and *what* the users interacting with it must do and *why*, is reduced to a minimum.

Tropos succeeds in its objectives by supporting five phases of software development:

Early requirements analysis. It concerns with the understanding of a problem by studying an existing organizational setting. 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 well or as easily— to achieve. The goals are achievable in virtue of reciprocal means-end knowledge and dependencies.

Late requirements analysis. In this phase, the system-to-be is described within its operational environment, along with relevant functions and qualities. This description models the system as a (relatively small) number of actors, which have a number of social dependencies with other actors in their environment.

Architectural design. It deals with the definition of the system global architecture in terms of subsystems, interconnected through data and control flows. In the Tropos framework, subsystems are represented as actors while data and control interconnections correspond to actor dependencies. In this step, actor capabilities are specified. This phase ends up with the specification of the system components, that may corresponds to software agents[2], but not necessarily. The component details are further analyzed in the following phase.

Detailed design. In this phase, each component or agent of the system architecture is defined in further details. In the case of the choice of an agent based architecture, the internal and external events, plans, beliefs, and agent communication protocols are specified. In other cases, the process still is applicable, provided some local modifications [13], although it is clear that the best advantages are obtained when AOP is addressed.

Implementation. The last phase concerns the actual implementation of the system in possibly, but not necessarily, AOP languages or paradigms, consistently with the detailed design.

[2]The choice of the programming paradigm is made only at this point in the methodology, with the advantage of minimizing the influences that this choices may have on the preceding development steps.

The five phases of the Tropos methodology has been motivated and illustrated with two case studies [9, 14, 10].

The present paper, instead, is mainly focused on the analysis of the early requirement phase and, partially, on the late requirement phase. In particular, concerning early requirements, the task of encoding initial informal requirement into the diagrammatic format used in the methodology, as well as the incremental transformational process that is at the basis of the construction of the complete model, will be addressed. With respect to previous papers [2], this revision mechanism will be motivated in its high level aspects and applied to a case study, rather than analyzed in its fundamental details. Another aspect that will be addressed, is the transition process from the early to the late requirements.

One of the goals of the Tropos project is to equip the methodology with an adequate tool which supports the software engineer during all the phases of software development, from early requirements down to the implementation. In particular, inside the IRST Tropos team, we are currently working at the definition of such a tool [1], called, since now on, the *Tropos tool*. This very *self motivating* case study will be used along all the paper.

The rest of the paper is structured as follows. Section 2 describes the Tropos tool problem and its early high level assumptions, section 3 shows how the early high level assumptions listed in Section 2 can be embodied into actor and goal diagrams, section 4 develops a preliminary actor diagram for late requirements. Conclusions and directions for further research are presented in section 5.

2 The problem

At the current state, the Tropos methodology is not provided with a tool which supports both the analyst along the process of acquiring, modeling, and analyzing requirements (the requirement engineering phases, namely, early requirement analysis and late requirement analysis) and the designer along the process of software engineering (namely, architectural design and detailed design). Of course, the availability of a tool that supports along the whole development process would be of a great impact on the applicability of the methodology. Although relevant efforts have already been made in order to provide a tool for managing diagrams in i^* [12], —the inspiring framework for Tropos early requirements, late requirements, and partially, architectural design— the realization of an integrated tool that supports all the phases of Tropos (or at least the first four) in a uniform and integrate way, has not been analyzed yet.

Early and late requirement analysis and architectural design in Tropos are aimed at producing a domain system conceptual and architectural model. This conceptual model must conform to the Tropos modeling language (described in terms of a UML metamodel) which offers actors, goals, and actor dependencies as primitive concepts for modeling. A Tropos model is defined as a set of instances of the meta-classes: *Actor*, *Goal*, *Softgoal*, *Plan*, *Resource*. The relationships among these meta-classes are instances of the reified *Dependency*, *Means-Ends analysis*, *Contribution* and *AND-OR decomposition*. A portion of the Tropos metamodel concerning the concept of actor and the notion of dependency [3] is shown in the class diagram of Figure 1.

For example, an actor dependency is a quaternary relationship, reified in the metamodel as the UML class *Dependency*. A dependency relates one *depender*, one *dependee*, and one *dependum*, and one optional *motivation*. The depender and the dependee are actors, while the dependum and the optional motivation may be goals, softgoals, plans or resources. For example, `PRG Manager`, in Figure 2, that is an instance of the meta-class *Position* of the Figure 1, is involved in a dependency relationship. In particular, `PRG Manager` (*depender*) depends on the `Architecture Designer` (*dependee*) in order to fulfill the goal `architectural design [$4] specified` (*dependum*); in this case, the *motivation* is not instancieted.

[3]The metamodel concerning the other concepts are defined analogously to the partial description reported here. A complete description of the Tropos modeling language can be found in [16, 17].

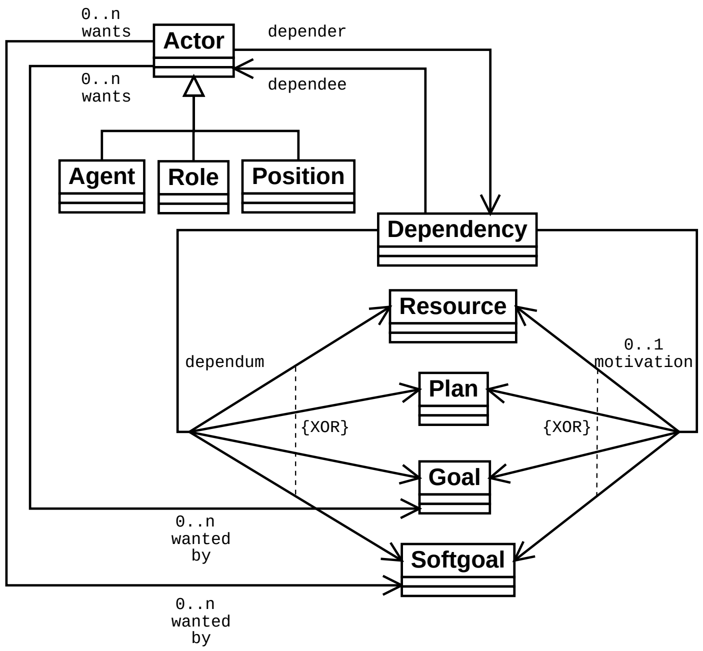

Figure 1: A portion of UML class diagram with respect to the meta concept of actor.

The meta-class definitions given in Figure 1, together with other five meta-class diagrams, provide for a full syntax of Tropos conceptual models [16]. Of course, the Tropos tool shall support for syntax checking during the conceptual model construction.

Another point to be taken under consideration is that, during the process of building actor and dependencies models, resulting from the analysis of social and system actors, the analyst may need to look at more diagrams from many perspectives at the same time. For example, she may want to look back why she introduced some subgoals and how she did it. Therefore, the Tropos tool must allow us to analyze the conceptual model from several points of view at the same time.

One important check that has to be performed during the analysis and before that it may be considered completed, is the closure of all the alternative or conjunctive decompositions (of goals, softgoals, and plans) and the evaluations of the goal and softgoal contributions. This process may be graphically visualized by putting and propagating ticks from the leafs up to the root (goal, softgoal, or plan) of the analysis tree, also using weighted propagation mechanisms for the qualitative (NFR) analysis [4]. This process will be later referred as the capability of managing ticks.

In addition, it is also foreseeable that the project manager wants to use a common graphics interface for viewing and analyzing documents, like feasibility study, requirements model, capability diagrams and so on, generated during different phases like the analysis phase and the design phase. Thus, it is desiderable that the Tropos tool adopts a common interface for different methodology phases, specially when adopting similar graphical notations, that is a pervasive characteristic of Tropos.

One of the main advantages of Tropos methodology is that the software engineer can also capture the *why* a sort of analysis is carried on or has been made. This important feature gives an extraordinary, although not fully explored yet, value to the notion of *traceability*. Traditionally, traceability is the property that a methodology or a CASE system exhibits when artifacts produced during later phases can be clearly referred back to artifacts or requirements produced earlier. In Tropos, this feature assumes an extra value due to two aspects. First, Tropos is aimed at uniformly covering development activities ranging from early requirements analysis down to actual implementation; thus, traceability in Tropos may be thought as spanning over several phases and very distant points in the development process. Second, the early and late requirements of Tropos provides an intentional analysis of the problem, facing the description of the *why*, together with that of the *what* and the *how*. In this context, tracing late artifacts back to early/late requirements provides a powerful method for giving strong motivations to all the developed artifacts, virtually, even to each single line of produced code, and justifying them with respect to the original requirements. Of course, the Tropos tool must give full support for traceability.

Finally, as always desired, also the Tropos tool has to be friendly, understandable and, above all, useful for the users like the analyst, the formal engineer, the designer, the developer and so on. Useful means that the user hat not to lose time in order to understand how the tool works, allowing her more time for depicting and investigating the best way to model the actors, their goals and the strategic dependencies.

In the following section, these preliminary requirements will be detailed and further analyzed, using the approach foreseen by the Tropos Early Requirement Analysis Phase.

3 Early Requirement Analysis

In Tropos, during the early requirement analysis, the requirement engineer models and analyzes the intentions of the stakeholders. The stakeholders intentions are modeled as goals that, through some form of analysis [3, 6], such as AND-OR decomposition, means-ends analysis, and contribution analysis, eventually lead to the functional and non-functional requirements of the system-to-be. To this end, *actor diagrams* for describing the network of social dependency relationships among actors, as well as *goal diagrams* for analyzing and trying to fulfill goals, are produced.

In our case study —the Tropos tool— some preliminary informal high level requirements has been collected, during a set of meetings and brainstormings of the IRST Tropos group.

The following list provides for a subset of the emerged requirements, and represents a further refinement with respect to the observations raised in section 2.

- Foundmental features of Tropos:
  - to support the software engineer along all the phases of the process software development (at least the first four);
  - to support the transformational approach as described in [2];
  - to allow both goal/plan analysis [3, 17] and non-functional analysis [4].
- System management:
  - to store a model and its views;
  - to manage multi-users which work together on the same model.
- Extensibility:
  - to integrate the Tropos tool with tools for formal analysis, like, for example, NuSMV [5, 15];
  - to integrate the Tropos tool with tools for managing the versioning, like, for example, CVS [7];
  - to integrate the Tropos tool with debugging tools.
- User interaction:
  - to visualize different possible views on the conceptual model, that may be needed both because of the different roles played by the user, and because of her changing design focus along the different phases;

  - to hide or expand subgraph in order to manage visual complexity;
  - to make multiple decomposition of the same goal from the perspective of one actor, thus allowing for more goal diagrams of the same goal;
  - to verify that the functional e non-functional (or quality) requirements are achieved (manage ticks).

- Help and documentation:
  - to give an understandable user guide of the Tropos language and concepts (ontology, metamodel for the informal language [16, 17], formal language [8, 15], and so on);
  - to generate documentation about a particular view on the model or a set of diagrams (i.e., static diagrams, dynamic diagrams);
  - to export the conceptual model or some views in several formats, like, for example, Scalable Vector Graphics (SVG) [18].

3.1 The analysis

After the definitions of the high level requirements as listed above, the process proceeds by identifying the most relevant stakeholders of the environment. In particular, the potential users of the tool, their goals, and the respective dependencies are identified, and then modeled by means of a first actor diagram.

In order to proceed with a deeper analysis, it is the case to recall that an actor may be embodied by the more specific notions of *agent*, *role* or *position*. An *agent* represents a concrete instantial actor like a *physical agent* (e.g., a person, a hardware system, and so on), or a *software agent*; a *role* corresponds to an abstract characterization of the behavior of an actor within some specialized context; a *position* represents a set of roles, typically played by one single agent. An agent can occupy a position, while a position is said to cover one or more roles. Also, an agent is said to play one or more roles. For more detailed distinctions and examples see [21].

In the first actor diagram depicted in Figure 2, `Software Professional` has been included as an agent who may occupy different positions: the `PRG Manager` , the `Analyst`, the `Formal Engineer`, the `Architecture designer`, the `Details designer` and the `Developer`.

Although in general it is assumed that each position covers several roles, in Figure 2, for lack of space, the roles covered by the various position are not reported, with the only exception of the role `Customer relations keeper`, given just as an example. The interesting aspect to be noticed here is that this role can be covered both by the `Analyst` position and by the `PRG Manager` position, that is, the agents that occupy one of these position, or possibly both, must play the role `Customer relations keeper`.

After the introduction of the actors, the dependency analysis starts by considering the actor `Customer`. In particular, she depends on the `PRG Manager` for achieving the goal `system provided`[4]. Also, the `PRG Manager` wants to manage the different versions of artifacts that other actors deliver to her (`manage versions and documentation`); in addition, she has the softgoal `artifacts [$1...$6] easily understandable`[5].

`PRG Manager` delegates two goals to the `Analyst`: `feasibility study [$1] delivered` and `requirements [$2] modeled`. The `PRG Manager` depends on the `Formal Engineer` for the goal `formal analysis [$3] delivered`; in order to

[4]Let's assume the convention of writing in passive predicative form the labels of goals and plans that can be delegated to other actors; instead, an active predicative form in the label will be used when delegation is not foreseen.

[5]Note that the place holders \$1. . . \$6 refer to objects mentioned in particular goals of the `PRG Manager`, namely `feasibility study [$1] delivered`, `requirements [$2] modeled`, `formal analysis [$3] delivered`, `architectural design [$4] specified`, `detailed design [$5] specified` and `code [$6] developed`.

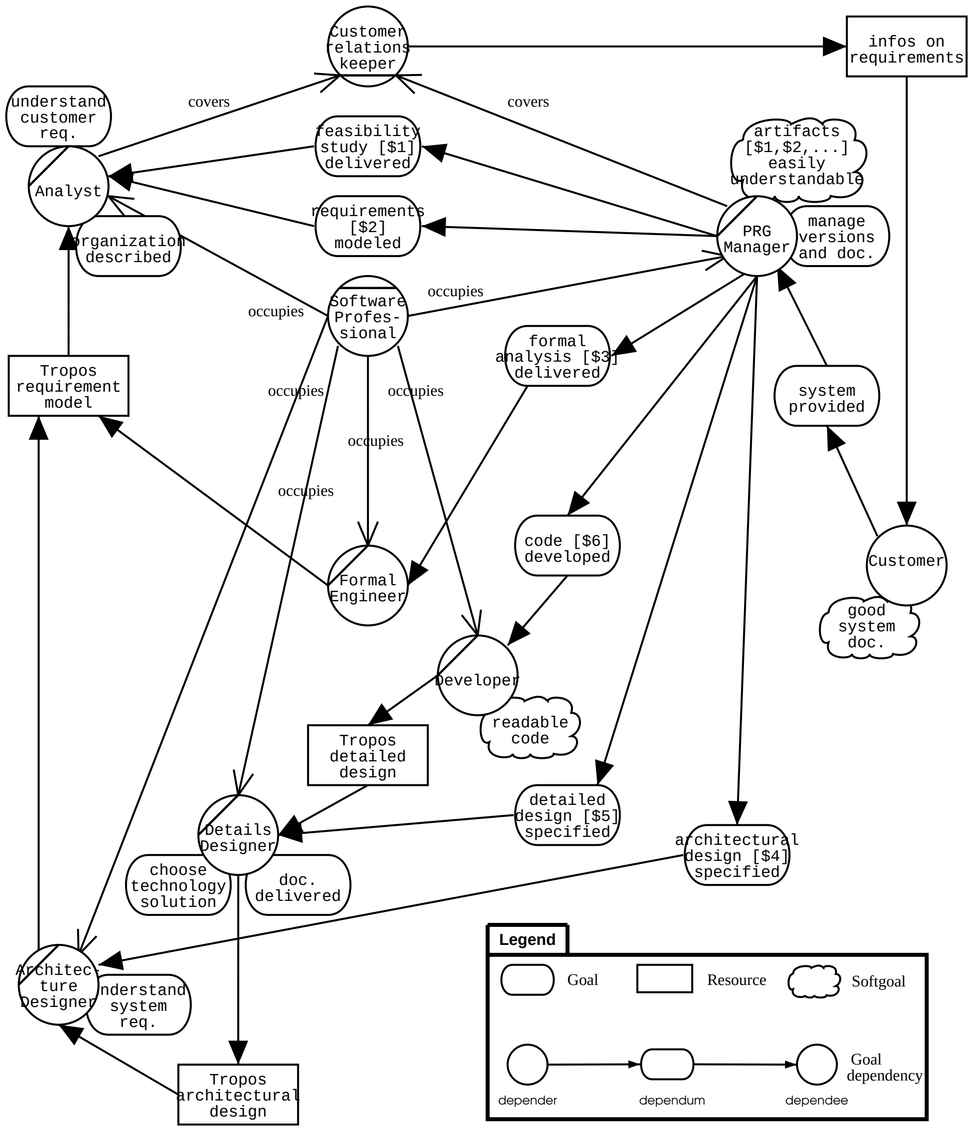

Figure 2: An actor diagram specifying the stakeholders of the Tropos tool project.

fulfill `formal analysis [$3] delivered` she relies on the resource `Tropos requirement model`. In a resource dependency, the depender depends on the dependee for the availability of an entity (physical or informational). Along a similar path, the `PRG Manager` depends on the `Architecture Designer` in order to achieve the `architectural design [$4] specified`, and she depends on the `Analyst` for the `Tropos requirement model`.

Another relevant position corresponds to the `Details Designer`, who has to produce a detailed design (`detailed design [$5] specified`); also, she depends on the `Architecture Designer` for the `Tropos architectural design`. Finally, `PRG Manager` depends on the `Developer` in order to develop the source code.

The next step in the analysis is the decomposition of each goal from the point of view of

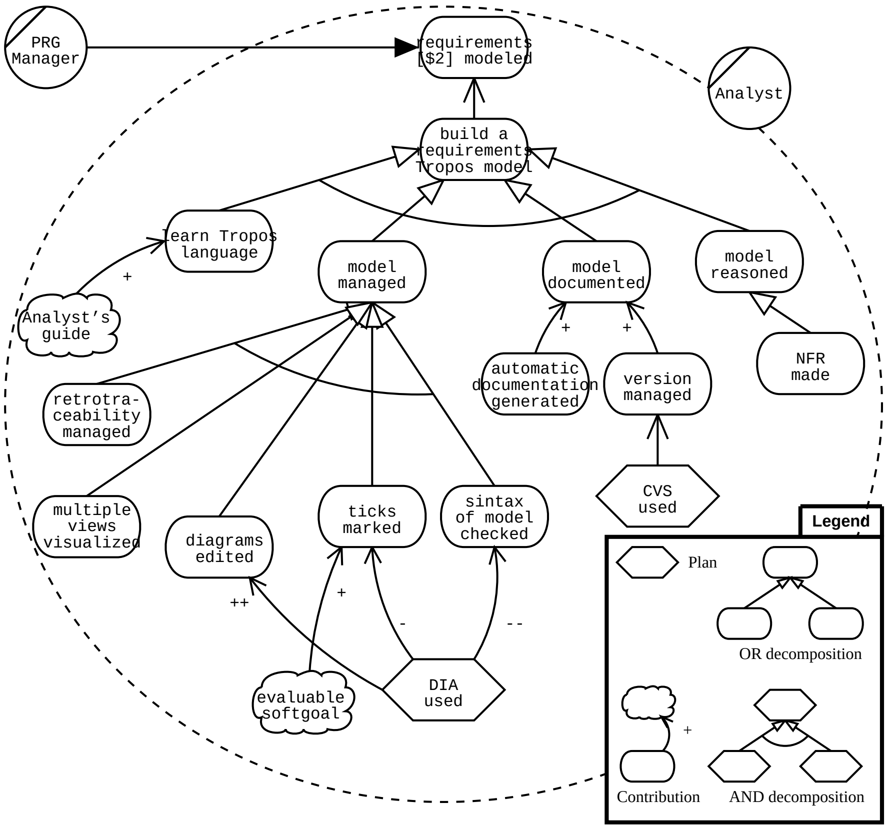

Figure 3: Goal analysis from the perspective of the `Analyst`.

the actor who committed for its fulfillment. Goals and plans (plan represents a set of actions which allows to satisfy one or more goals) are analyzed from the perspective of each specific actor in turn, by using three basic analysis techniques: *means-ends analysis*, *contribution analysis*, and *AND-OR decomposition* [4, 17]. For goals, means-ends analysis proceeds by refining a goal into subgoals in order to identify goals, plans, resources and softgoals that provide means for achieving the goal (the end). Contribution analysis allows the designer to point out goals, softgoals, and plans that can contribute positively or negatively at reaching the goal under analysis. AND-OR decomposition allows for a combination of AND and OR decompositions of a root goal into subgoals, thereby refining a goal structure.

As depicted in Figure 3, the goal `requirements [$2] modeled` delegated from the `PRG Manager` to the `Analyst` is the "end" in a means-ends analysis; of course, the chosen "mean" is the goal `build a requirements Tropos model`, that make explicit our implicit "strong" requirement of building a Tropos oriented tool. The goal `build a requirement Tropos model` is then AND decomposed into the four subgoals: `learn Tropos language`, `model managed`, `model documented` and `model reasoned`.

The softgoal `Analyst's guide`, contributes positively at satisfying the goal `learn Tropos language`. In order to build a requirements Tropos model (`model managed`), it is indispensable to create a new model, manage an exist model, add/edit/remove some elements of the model and so on. All this corresponds to the goal `diagram edited`[6]. In addition, also some goals concerning the most important features of Tropos, already intro-

[6]Further decomposition of this and other goals could be provided, but, of course, here we have to limit the size of our presentation. See [1] for further details.

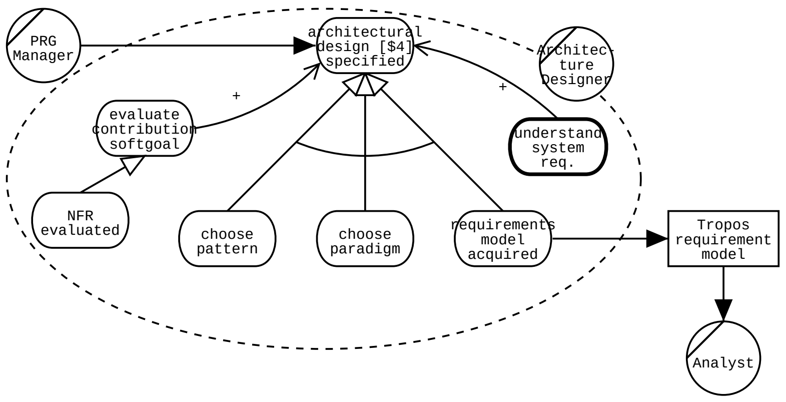

Figure 4: A goal diagram including the `Architecture Designer`.

duced in section 3, are here depicted, that are: `retrotraceability managed`, in order to deal with backward and multiple phases traceability, `ticks marked`, in order to deal with the ticks mechanism for the check of the completeness of model analysis, and `syntax of model checked`, to verify the syntax of the model with respect to the metamodel as described in [16, 17]. Finally, also multiple views management is considered (`multiple views visualized`).

For the ticks mechanism, it is useful to take into account the softgoals contribution, represented by the softgoal `evaluable softgoal`. A first proposal for partially fulfilling `model managed` relied on the use of DIA[7] graphic tool (`DIA used`). Although this solution could satisfy `diagrams edited` (note the positive contribution $++$), it lacks to satisfy `ticks marked` and `syntax of model checked` (note the negative contribution $-$ and $--$, respectively).

The subgoals `automatic documentation generated` and `version managed` contribute positively at the achievement of the goal `model documented`. The first subgoal concerns with the automatic generation of the documentation about the conceptual model and/or some views on it, while the second regards the management of different versions for each analyst. A mean for the fulfillment of the last subgoal is to integrate the CVS software system in the Tropos tool. Finally, the last subgoal `model reasoned` can be satisfied with a *Non-Functional Analysis*, as proposed in [4] (`NFR made`).

Figure 4 shows the goal diagram from the point of view of the position `Architecture Designer`. The goal delegated by `PRG Manager` (see Figure 2) is labelled with `architectural design [$4] specified`. This is AND decomposed into three subgoals: `choose pattern`, `choose paradigm` and `requirements model acquired`. This last subgoal may be considered as depending on the resource `Tropos requirements model` that, by further analysis, can be assumed as to be delegated to the `Analyst`. `choose paradigm` concerns with the selection among several architectural paradigms (e.g., BDI architecture). Aside the AND decomposition, the root goal `architectural design [$4] specified` also receives positive contributions from two goals: `understand system requirements` and `evaluate contribution softgoal`. The first one was originally foreseen as a proper goal of `Architecture Designer` (see Figure 2). It has been chosen here, as well as in an analogous situations in other Figures, to evidence this fact by using thick outlines. The second contributing goal is further enabled by `NFR evaluated`.

The next goal diagram, depicted in Figure 5, refers to a goal decomposition of the `detailed design [$5] specified`, from the perspective of the dependee `Details`

[7]Dia is a graphic tool that supports many language like UML, ER, and so on [11].

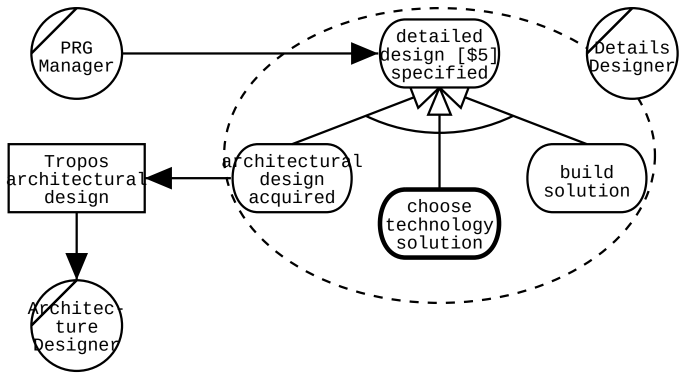
PRG Manager
detailed design [$5] specified
Details Designer
Tropos architectural design
architectural design acquired
choose technology solution
build solution
Architecture Designer

Figure 5: A goal diagram including the `Details Designer`.

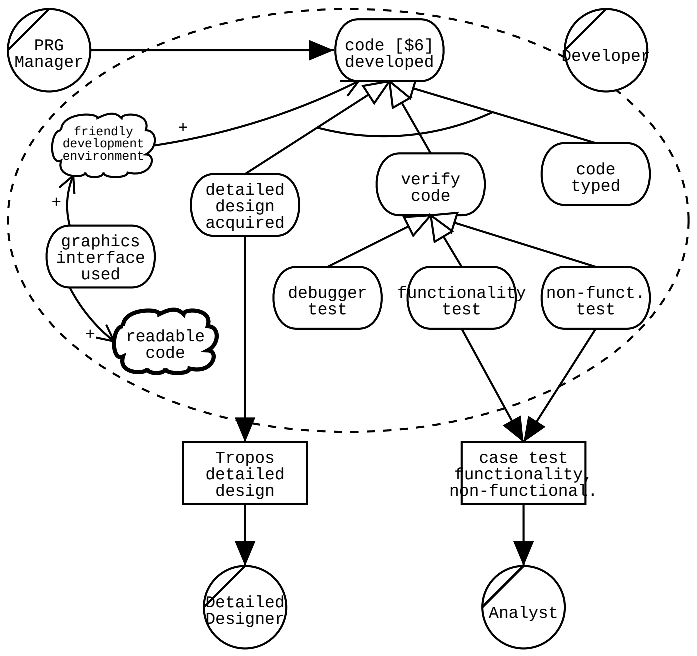

Figure 6: Goal diagram from the point of view of the `Developer`.

`Designer`. It is split in further subgoals (AND decomposition). In particular, it is relevant here to underline that the goal `choose technology solution`, which was originally introduced in Figure 2 as a goal of the position `Details Designer`. Finally, the designer has to implement the solutions (`build solution`) using the formalism foreseen by the Tropos methodology, for the detailed design phase.

Figure 6 represents the decomposition of the goal `code [$6] developed`. This receives a partially positive contribution from the softgoal `friendly development environment`, which is contributed by the goal `graphics interface used`: a further contribution is for the softgoal `readable code`. `code [$6] developed` is also AND-decomposed into the three subgoals: `detailed design acquired`, `verify code`, and `code typed`. The position `Developer` can achieve one of the three subgoals (OR de-

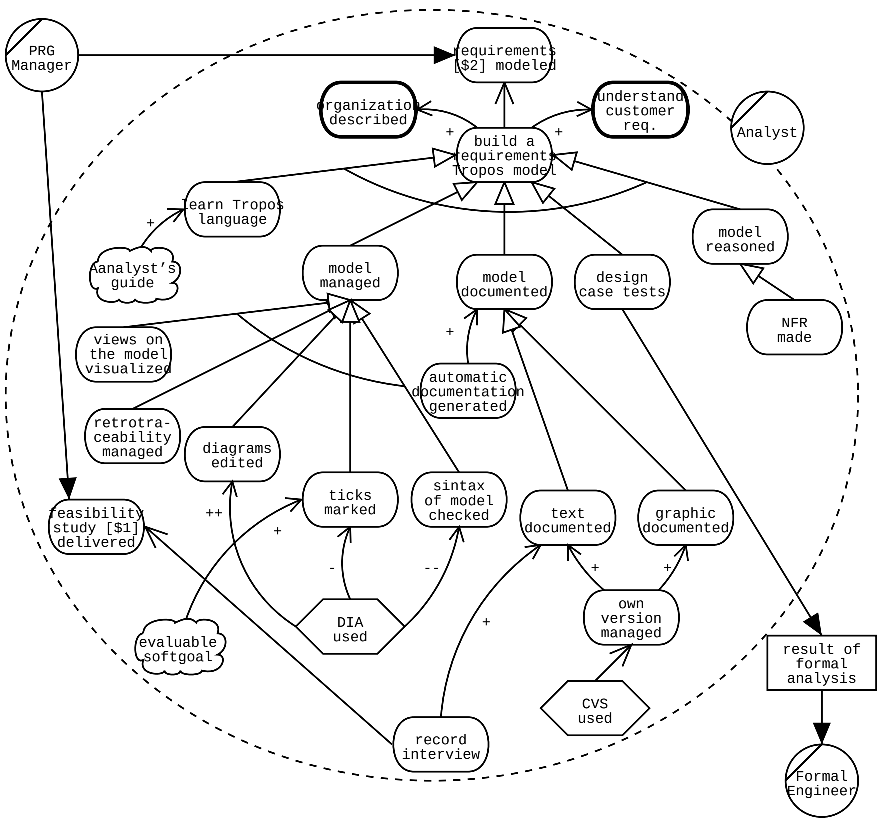

Figure 7: Revising Goal analysis from the point of view of the `Analyst`.

composition): `debugger test`, `functionality test`, and `non-functionality test`, in order to satisfy `verify code`.

3.2 Revising the analysis

The previous section presented a first result of the activity of requirement definition and analysis. It is obvious that the diagrams developed are not sufficiently detailed. For this reason, iterative steps of incremental refinement of the model have to be performed. As already mentioned in the introduction, this way of proceeding is a typical feature of the Tropos methodology. The final setting of each Tropos phase may be reached after possibly several refinements, in each one of which not only new details may be added, but, also, already present elements and dependencies can be revised or even deleted [2]. This iterative process not only may require intra-phase refinements, but, possibly, also revisions of artifacts produced during early phases (inter-phases refinements). The importance of retrotraceability is, here, evident.

Just as an example of the intra-phase refinement activity, the revision of the goal diagram in Figure 3 is presented in Figure 7.

The contributions from `build a requirements Tropos model` to `organization described` and `understand customer requirements` are the first relevant differences. The two positively contributed goals, `organization described` and `understand customer requirements`, were initially considered as original `Analyst`'s goal (see Figure 2), but later not further analyzed (in fact, they are not taken into account in Figure 3), until the revision now proposed. Thus, this revision is necessary to complete the analysis of the requirements initially introduced in Figure 2.

Among others new elements added by the revision, let's note the subgoal `design case tests`, considered necessary because the analyst has also to design the cases of test in order to validate and verify the functionalities (under form of functional and non-functional requirements) of the software system. Another source of revision derives from the observation that the evaluation of the formal analysis conducted by `Formal Engineer` can provide more hints to the `Analyst`; thus, Figure 7 introduces also a dependence among the `Analyst` and the `Formal Engineer`, upon the resource `result of formal analysis`, motivated by the fulfillment of `design case tests`. Another softgoal of `build a requirements Tropos model`, namely `model documented`, has been OR decomposed into `text documented` and `graphic documented`. Finally, the goal `feasibility study [$1] delivered` (see Figure 2), became the end in a means-ends analysis where the "mean" is the goal `record interview`, which, also, contributes positively for the fulfillment of the `text documented`.

4 Late requirements

During late requirement analysis the system-to-be (the Tropos tool in our case) is described within its operating environment, along with relevant functions and qualities. The system is represented as one actor which have a number of dependencies with the other actors of the organization. These dependencies define all the functional and the non-functional requirements for the system-to-be.

A very important feature of Tropos is that the kind of representations adopted for the early requirements (that are, the actor and the goal diagrams) are used in the same way also during the late requirement analysis. The only different element is that, here, the system-to-be is introduced in the analysis as one of the actors of the global environment. Of course, the goal of the engineer, since now on, is to decompose and analyze in details the system goals. To do this a sequence of more and more precise and refined goal diagrams for the actor `Tropos tool`, in our case, have to be produced, applying the iterative refinement process already introduced in the previous section.

During the goal analysis of the system-to-be, of course, some differences, with respect to the early analysis, may be taken into account, as, for example, the fact that the system-to-be is characterized by a much different level of intentionality and autonomy if compared with the social actors (it may be assumed that the system is much more prone to fulfill the delegated goals and tasks —it is built for this— than a social actor, and that it has no back dependency on the social environment, unless for resources representing I/O flow)[8]. Also, the analysis of the system goals should be carried on inside the scope of the system as far as possible, trying to minimize revisions of decisions already stated during early requirements analysis, although, in some cases, this hypothesis may happen. Finally, the system-to-be is characterized only by delegated goals, that are, thus, exclusively generated, in this simplified setting, by the early requirement analysis.

Due to lack of space, and considering, apart the principled differences listed above, that there is no technical difference in the process of late requirement analysis with respect to the process of early requirement analysis, only a short description of the first steps is here introduced. In particular, it is relevant to show the very beginning of late requirement analysis, and how a set of goal can be assigned to the system-to-be starting from the previous analyses. In fact, as mentioned above, the system goals, softgoals and plans have to be motivated by the unresolved goals, softgoals and plans elicited in the early requirements. At is end, Figure 8 highlights which goals, softgoals and plans in the social actors goal diagrams may be satisfied by means of strategic dependencies on the `Tropos tool`. It is worth noticing that not

[8]Some more outgoing dependencies can be, really, foreseen, specially if we consider the system in relations with other systems or its subsystems. This possibility, indeed, allow us for an intersecting level of analysis for, e.g., the so called *autonomous systems*. In any case, the choice of relate the system with other systems, or decompose it into subsystems, may come later in the process, and fall out of the scope of the present paper.

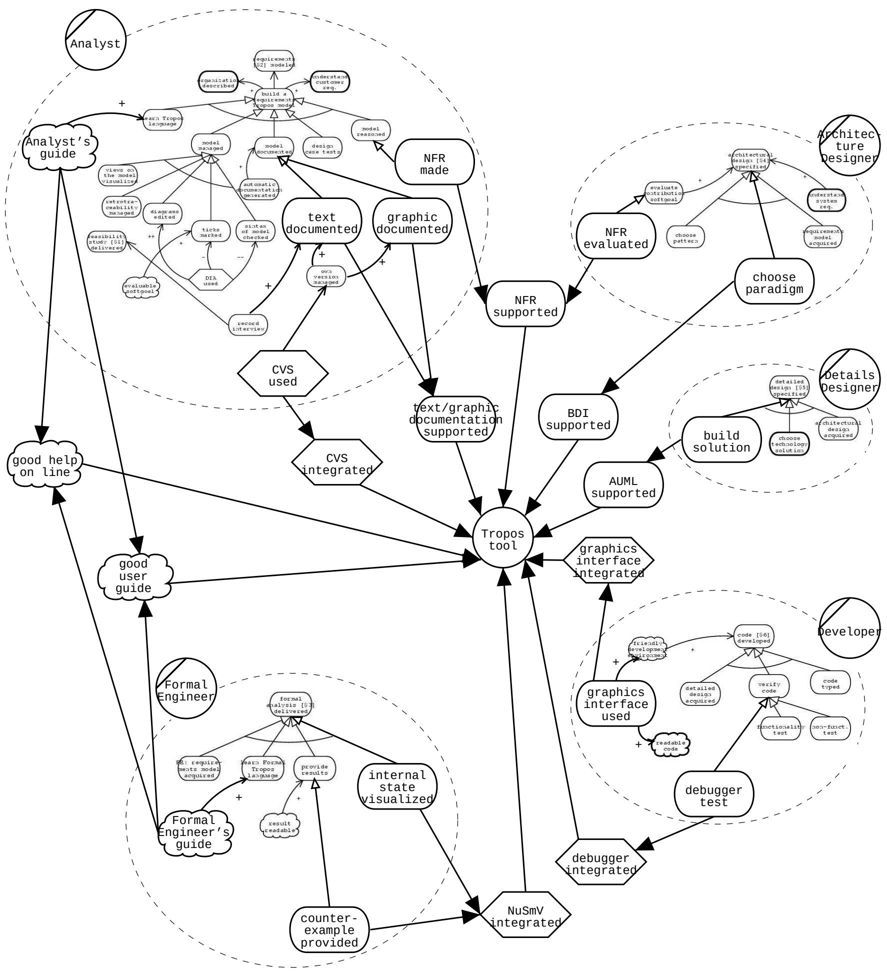

Figure 8: Actor diagram: focus on the system actor `Tropos tool`.

necessarily an unresolved social actor goal has to be resolved through a goal dependency on the system, but it may generate, instead, for example, a plan dependency, as happens for `debugger integrated`. The same may apply to all the kinds of dependum.

Referring in detail at Figure 8, it can be seen that `Analyst` depends on the `Tropos tool` for `NFR supported`, as well as the `Architecture Designer` does, although for a different goal. `Analyst` also depends on the goal `text/graphic documentation supported` in order to contribute to `text documented` and to `graphic documented`, on the plan `CVS integrated` in order to fulfill the plan `CVS used`, finally, and on the softgoals `good user guide` and `good help on line`. As another example, let's consider the position `Formal Engineer`, whose diagram had not been presented in the previous pages due to lack of space. She depends on the `Tropos tool` for the plans `NuSMV integrated`, with the motivations of `internal state visualized` and `counter-example provided`. Other example, not commented here, can be seen in Figure 8, that, of course, has the only aim of exemplifying the process, and has not to be considered exhaustive.

5 Conclusion

In the present paper the definition process of a Tropos tool, that currently is in phase of analysis and design at IRST, has been used as a case study for presenting some features of the Tropos methodology itself.

First of all, a stronger emphasis, with respect to previous papers [9, 14, 10], has been put on the process that lead from the very informal elicitations of requirements to their descriptions in terms of actor and goal diagrams. Also, the goal definition, analysis, and revision process has been presented, pointing out, in particular, how the goal diagram construction has to be considered as an incremental process, running through a sequence of revision steps.

Finally, the last figure shown in the paper is aimed at illustrating how the transition from the early requirement analysis to the late requirement analysis can be seen as part of a smooth and natural process.

The management of traceability has been raised as a crucial point for correctly dealing with the revision (specially the inter-phase revision) process. In future works, we aim at further develop this issue with other specifically focused case studies and examples. Nevertheless, we believe that clearly showing the connecting items (dependums) between early and late requirements, as done in Figure 8, already provides for a first step into the solution. Of course the best support can be reached only through the development of an appropriate tool: the Tropos tool.

6 Acknowledgments

Our acknowledgments are addressed to all the IRST people working at the definition of the Tropos tool. A special thanks is due to Eric Yu, who provided precious suggestions and comments during very preliminary discussions on the idea.

References

[1] D. Bertolini, P. Bresciani, A. Daprà, A. Perini, and F. Sannicolò. The Tropos tool: analysis and design. Technical Report 0203-01, ITC-IRST, via Sommarive, Povo, Trento, January 2002.

[2] P. Bresciani, A. Perini, P. Giorgini, F. Giunchiglia, and J. Mylopoulos. Modelling early requirements in Tropos: a transformation based approach. In *Second International Workshop on Agent-Oriented Software Engineering (AOSE-2001)*, Montreal, Canada, May29th 2001.

[3] L. Chung, S. Liao, W. Huaiqing, E. Yu, and J. Mylopoulos. Exploring alternatives during requirements analysis. *IEEE Software*, 18(1), jan, feb 2001.

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

[5] A. Cimatti, E. M. Clarke, F. Giunchiglia, and M. Roveri. NuSMV: a new symbolic model checker. *International Journal on Software Tools for Technology Transfer (STTT)*, 2(4), March 2000.

[6] A. Dardenne, A. van Lamsweerde, and S. Fickas. Goal-directed requirements acquisition. *Science of Computer Programming*, 20(1-2):3–50, 1993.

[7] P. Cederqvist et al. *Version Management with CVS*. http://www.cvshome.org/docs/manual/.

[8] A. Fuxman. *Formal analysis of early requirements specifications*. PhD thesis, University of Toronto, Toronto, Canada, 2001. Capitolo 3.

[9] P. Giorgini, A. Perini, J. Mylopoulos, F. Giunchiglia, and P. Bresciani. Agent-oriented software development: A case study. In S. Sen J.P. Müller, E. Andre and C. Frassen, editors, *Proceedings of the Thirteenth International Conference on Software Engineering - Knowledge Engineering (SEKE01)*, Buenos Aires - ARGENTINA, June 13 - 15 2001.

[10] F. Giunchiglia, A. Perini, and F. Sannicolò. Knowledge level software engineering. In Springer Verlag, editor, *In Proceedings of ATAL 2001, Seattle*, December 2001. Also IRST Technical Report 0112-22, Istituto Trentino di Cultura, Trento,Italy.

[11] Gnome. *Dia Tutorial*. http://www.lysator.liu.se/∼alla/dia/diatut/all/all.html.

[12] Knowledge Management Lab at the University of Toronto. *OME3 Documentation*. http://www.cs.toronto.edu/km/ome/docs/manual/manual.html.

[13] A. Perini, P. Bresciani, P. Giorgini, F. Giunchiglia, and J. Mylopoulos. Towards an Agent Oriented approach to Software Engineering. In *Proceedings of the Workshop. Dagli oggetti agli agenti: tendenze evolutive dei sistemi software*, Modena, Italy, 4 - 5 Sept 2001.

[14] 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 CA, 28 May - 1 June 2001.

[15] M. Pistore, A. Fuxman, J. Mylopoulos, and P. Traverso. Model Checking Early Requirements Specifications in Tropos. In *Proceedings Fifth IEEE International Symposium on Requirements Engineering (RE01)*, Toronto, Canada, August 27-31 2001.

[16] F. Sannicolò. Tropos: una Metodologia ed un Linguaggio di Modellazione Visuale Semiformale. Master's thesis, Università degli Studi di Trento - Facoltà di Scienze Matematiche Fisiche e Naturali, via Sommarive, Povo, Trento, December 2001. Also IRST Technical Report 0201-01, Istituto Trentino di Cultura, Trento,Italy.

[17] F. Sannicolò, A. Perini, and F. Giunchiglia. The Tropos modeling language. A User Guide. Technical Report 0202-12, ITC-IRST, Jan 2002.

[18] W3C. *Scalable Vector Graphics (SVG) 1.0 Specification*, September 2001. http://www.w3.org/TR/2001/REC-SVG-20010904/REC-SVG-20010904.pdf.

[19] E. Yu. Modeling organizations for information systems requirements engineering. In *Proceedings First IEEE International Symposium on Requirements Engineering*, pages 34–41, San Jose, January 1993. IEEE.

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

[21] E. Yu. Software Versus the World. In *Agent-Oriented Software Engineering AOSE-2001 Workshop Proceedings*, LNCS 2222, Montreal, Canada, 29 May 2001.

[22] E. Yu and J. Mylopoulos. Understanding 'why' in software process modeling, analysis and design. In *Proceedings Sixteenth International Conference on Software Engineering*, pages 159–168, Sorrento, Italy, May 1994.