Requirements Analysis in Tropos: a self referencing example

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 heavily characterized, among other features, by the fact that it pays great 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 is obtained by means of the two requirement phases: the *early requirements analysis* and the *late requirements analysis*. Moreover, Tropos uses, along these phases, a uniform notation and an homogeneous, smooth, incremental, and iterative process, based on a set of progressive transformational steps.
This 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. The focus here is on the early requirements and on how to manage the transition from them to the late requirement analysis.

1 Introduction

Tropos [19, 13, 11] is a novel agent-oriented software engineering methodology characterized by three key aspects [18]. First, it pays attention to the activities that precede the specification of the prescriptive requirements, like understanding *how* and *why* the intended system would meet the organizational goals[1]. Second, it deals with all the phases of system requirement analysis and all the phases of system design and implementation in a uniform and homogeneous way, based on common mentalistic notions as those of *actors*, *goals*, *softgoals*, *plans*, *resources*, and *intentional dependencies*. Third, the 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, by means of a sequence of transformational steps [3].

One of the main advantages of the Tropos methodology is that it allows to 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 its non-functional requirements. Tropos, although not exclusively, addresses particularly well the Agent Oriented Programming [18]. In fact, the decision of using mentalistic notions in all the analysis phases has important consequences. In particular,

[1] In this, Tropos is largely inspired by Eric Yu's framework for requirements engineering, called i^*, which offers actors, goals, and actor dependencies as primitive concepts [23, 24, 26].

agent oriented specifications and programs use the same notions and abstractions used to describe the behavior of human agents and the processes involving them; thus, the conceptual gap between users' specifications (in terms of *why* and *what*) and system realization (in terms of *what* and *how*), is reduced to a minimum.

Tropos supports five phases of software development. The **early requirements analysis** is concerned 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. During the **late requirements analysis**, 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. The **architectural design** phase deals with the definition of the system global architecture in terms of subsystems, that are represented as actors, and their data dependencies, that are represented as actor dependencies. The **detailed design** phase aims at specifying each architectural component in further detail (adopting a subset of the AUML diagrams [17, 1]) in terms of inputs, outputs, control and other relevant information. Finally, during the **implementation** phase, the actual implementation of the system is carried out, consistently with the detailed design. More details and motivation about these five phases can be found in [12, 11, 13, 19].

The present paper mainly focuses on the analysis of the early requirement analysis phase and, partially, on the late requirement analysis phase. In particular, concerning the early requirements analysis, the task of encoding initial informal requirements 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 [3], 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 in this paper is the transition process from the early to the late requirements. The example that will be used along all the present paper to illustrate the above aspects is the definition of a support tool for the Tropos methodology itself, called, since now on, the *Tropos tool*.

The rest of the paper is structured as follows. Section 2 describes the Tropos tool problem and some background motivating some early choices. Section 3 shows how these choices can be embodied into actor and goal diagrams. Section 4 develops a preliminary actor diagram for late requirements with the only aim to provide a glance on the transition from early to late requirements. Conclusions and directions for further research are presented in Section 5.

2 The problem

As of today, the Tropos methodology lacks a tool which supports both the analyst during the process of acquiring, modeling, and analyzing requirements and the designer during the process of software engineering. Of course, the availability of a tool that supports along the whole development process would be of 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^* [15], 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.

In order to start with the definition of the Tropos tool, some observations may be useful. First, let us recall that early and late requirement analysis and architectural design in Tropos are aimed at producing domain and system conceptual and architectural models. These conceptual models must conform to a precise Tropos modeling language [20]. 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 shall 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 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 [6]. This process will be later referred as the capability of *managing ticks*.

In addition, it may also be the case that the project manager wants to use a common graphical interface for viewing and analyzing documents (feasibility study, requirements model, capability diagrams, and so on) generated during different phases, such as the analysis phase and the design phase. Thus, it is desirable that the Tropos tool adopts a common interface for different phases of the methodology, specially when adopting similar graphical notations, which is a pervasive characteristic of Tropos.

One of the main advantages of the 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 yet fully explored, 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 the 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 requirements —both early and late— provides a powerful method for giving strong motivations to all the developed artifacts, virtually even to each single line of the 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, the Tropos tool has also to be friendly, understandable and, above all, useful for all the users, like the analyst, the formal engineer, the designer, the developer, and so on. Useful means that the user has not to waste time in order to understand how the tool works, giving her more time for depicting and investigating the best way to model the actors, their goals, and their strategic dependencies.

In the following section, these preliminary requirements will be detailed and further analyzed, by means of an early requirement analysis.

3 Early Requirement Analysis

In Tropos, during the early requirement analysis, the requirement engineer models and analyzes the intentions of the stakeholders. These are modeled as goals that, through some form of analysis [16, 8], 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.

Starting from our common understanding of the Tropos tool problem, as described in the previous section, the following list of more detailed issues to be addressed may be provided:

- Fundamental features of Tropos:
  - software engineer support for all the phases of the software development process (at least the first four);
  - support for the transformational approach as in [3];
  - goal/plan analysis [16, 8] and non-functional analysis [6].
- System management:
  - model and views storage;
  - multi-users management.
- Extensibility:
  - integration with tools for formal analysis (e.g., NuSMV [10, 7]) and version management (e.g., CVS [9]);
  - integration with debugging tools.
- User interaction:
  - visualization of different possible views on the conceptual model, which may be needed both because of the different roles played by the user, and because of her changing design focus along the different phases;
  - subgraph visualization and zooming in order to manage visual complexity;
  - support for multiple decomposition of the same goal from the perspective of one actor, thus allowing for more goal diagrams of the same goal;
  - verification that the functional and non-functional (or quality) requirements are achieved (manage ticks).
- Help and documentation:

- user guide for the Tropos language and its concepts (ontology, metamodel for the informal language [20], formal language [10], and so on);
- automatic generation of the documentation about a particular view on the model or a set of diagrams (i.e., static diagrams, dynamic diagrams);
- export of the conceptual model or some views in several formats (e.g., Scalable Vector Graphics (SVG) [21]).

3.1 The analysis

After the definitions of the high level requirements, 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 worth to recall that an actor may be embodied by the more specific notions of *agent*, *role* and *position*. An *agent* represents a concrete instantial actor like a *physical agent* (e.g., a person, an hardware system, and so on), or a *software agent*; a *role* corresponds to an abstract characterization of one specific behavior of an actor within some specialized context; a *position* represents a set of roles, typically played by a 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 [24, 25].

In the actor diagram depicted in Figure 1, `Software Professional` is depicted as an agent who may occupy different positions: `PRG Manager` , `Analyst`, `Formal Engineer`, `Architecture Designer`, `Details Designer`, and `Developer`.

Although in general it is assumed that each position covers several roles, in Figure 1, for lack of space, only one role is present as an example: `Customer relations keeper`. 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, each agent that occupies one of these position, or possibly both, must play the role `Customer relations keeper`.

After the introduction of the actors, the dependency analysis focuses on the dependencies of each single different actor. For example, `Customer` depends on the `PRG Manager` for achieving the goal `system provided`. Also, the `PRG Manager` wants to manage the different versions of artifacts that other actors deliver to her (`manage versions and doc.`); in addition, she has the softgoal `artifacts [$1...$6] easily understandable`[2].

`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 fulfill `formal analysis [$3] delivered` the `Formal Engineer` relies on the `Analyst` for the resource `Tropos requirement model`. In a resource dependency, the depender depends on the dependee for the availability of a physical

[2] 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`.

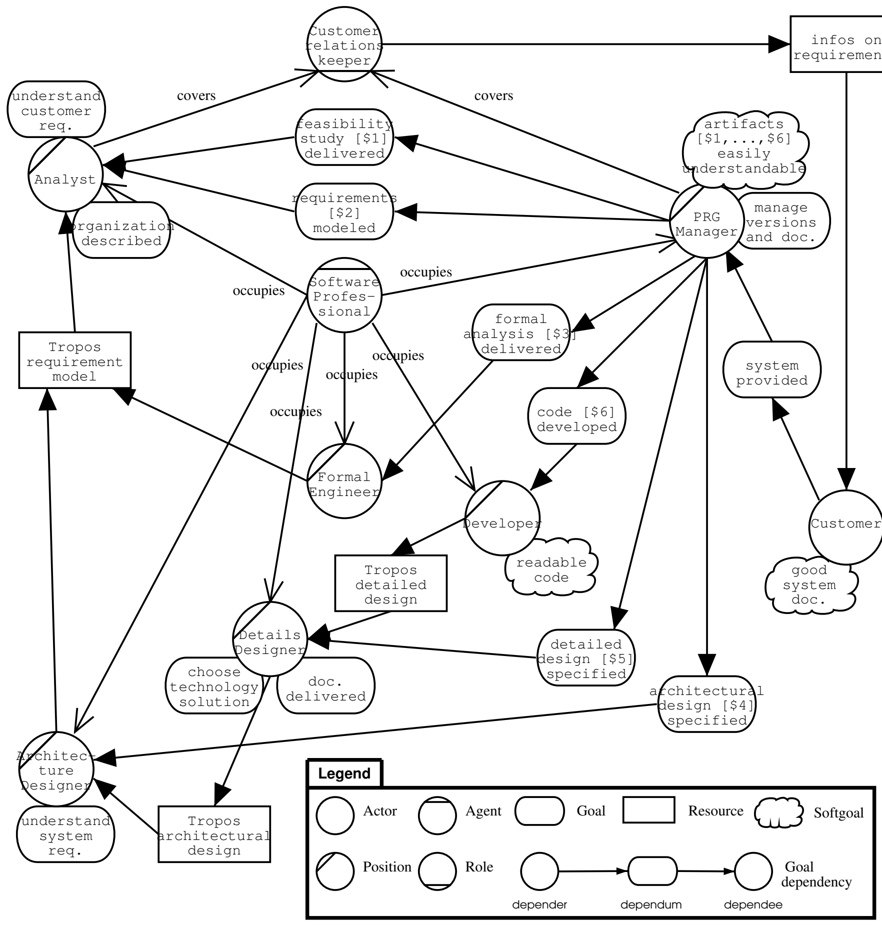

Fig. 1. An actor diagram specifying the stakeholders of the Tropos tool project.

or informational entity (dependum). Along a similar path, the `PRG Manager` depends on the `Architecture Designer` for achieving the `architectural design [$4] specified`, and `Architecture Designer` 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 (`code [$6] developed`).

The next step in the analysis is the decomposition of each goal from the point of view of the actor who committed for its fulfillment. Goals and plans (plan represents a set of actions which allow for the satisfaction of 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* [16,

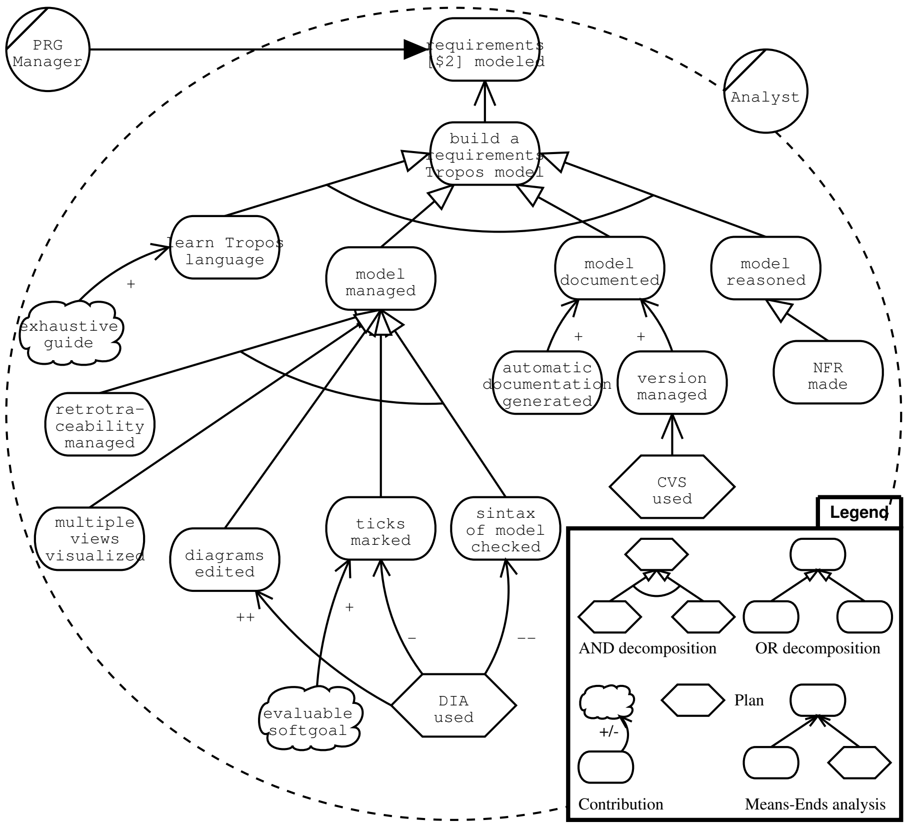

Fig. 2. Goal analysis from the perspective of the `Analyst`.

6, 20]. For goals, means-ends analysis aims at identifying 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 2, 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 makes 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 `exhaustive 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

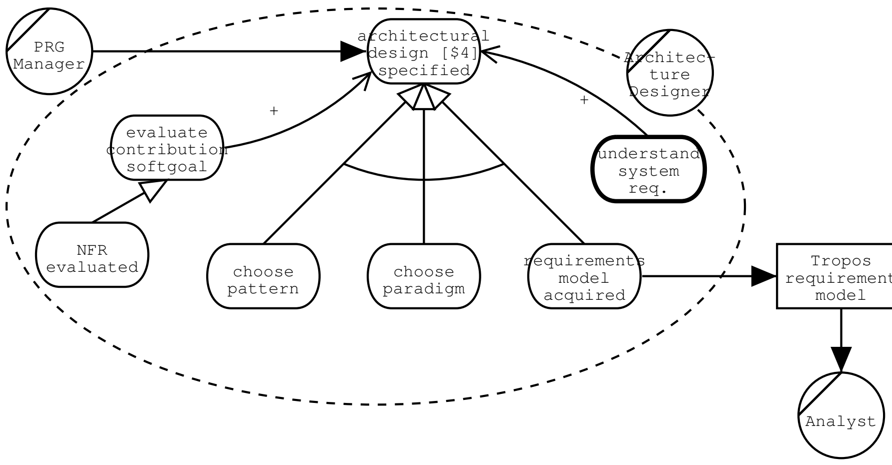

Fig. 3. A goal diagram including the `Architecture Designer`.

elements of the model and so on. All this corresponds to the goal `diagrams edited`[3]. In addition, some goals concerning the most important features of Tropos, already introduced in Section 3, are also depicted here, 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 [20]. 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[4] graphic tool (`DIA used`). Although this solution could satisfy `diagrams edited` (note the positive contribution ++), it does not satisfy `ticks marked` and `syntax of model checked` (note the negative contribution $-$ and $--$, respectively).

The subgoals `automatic documentation generated` and `version managed` contribute positively to 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 [6] (`NFR made`).

Figure 3 shows the goal diagram from the point of view of the position `Architecture Designer`. The goal delegated by `PRG Manager` (see Figure 1) is

[3] Further decomposition of this and other goals are not provided because of space limits. See [5, 2] for further details.

[4] Dia is a graphic tool that supports many language like UML, ER, and so on [14].

labeled 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 goal was originally foreseen as a proper goal of `Architecture Designer` (see Figure 1). This fact is evidenced here, as well as in analogous situations in other figures, by using a thicker borderline.

The second contributing goal is further enabled by `NFR evaluated`. Due to lack of space, the goal diagrams of `Details Design`, `Formal Engineer`, and `Developer` are not reported here, but can be found in [5, 2].

3.2 Revising the analysis

The previous section presented a first result of the activity of requirements 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 possibly after several refinements, in each of which not only new details may be added, but, also, already present elements and dependencies can be revised or even deleted [3]. This iterative process not only may require intra-phase refinements, but, possibly, also revisions of artifacts produced during earlier 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 2 is presented in Figure 4.

The positive contributions from `build a requirements Tropos model` to the goals `organization described` and `understand customer requirements` are the first relevant differences. The two positively contributed goals —namely, `organization described` and `understand customer requirements`— were initially considered as original `Analyst`'s goal (see Figure 1), but later not further analyzed (in fact, they are not taken into account in Figure 2), until the revision depicted in Figure 4. Thus, this revision is necessary to complete the analysis of the requirements initially introduced in Figure 1.

Among other new elements added by the revision, let us note the subgoal `design case tests`, considered necessary because the analyst has also to design the cases to be tested in order to validate and verify the functionalities of the software system with respect to the functional and non-functional requirements. 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 4 also introduces a dependence between the `Analyst` and the `Formal Engineer`, upon the resource `result of formal analysis`, motivated by the fulfillment of `design case tests`. Another subgoal of `build a requirements Tropos model`,

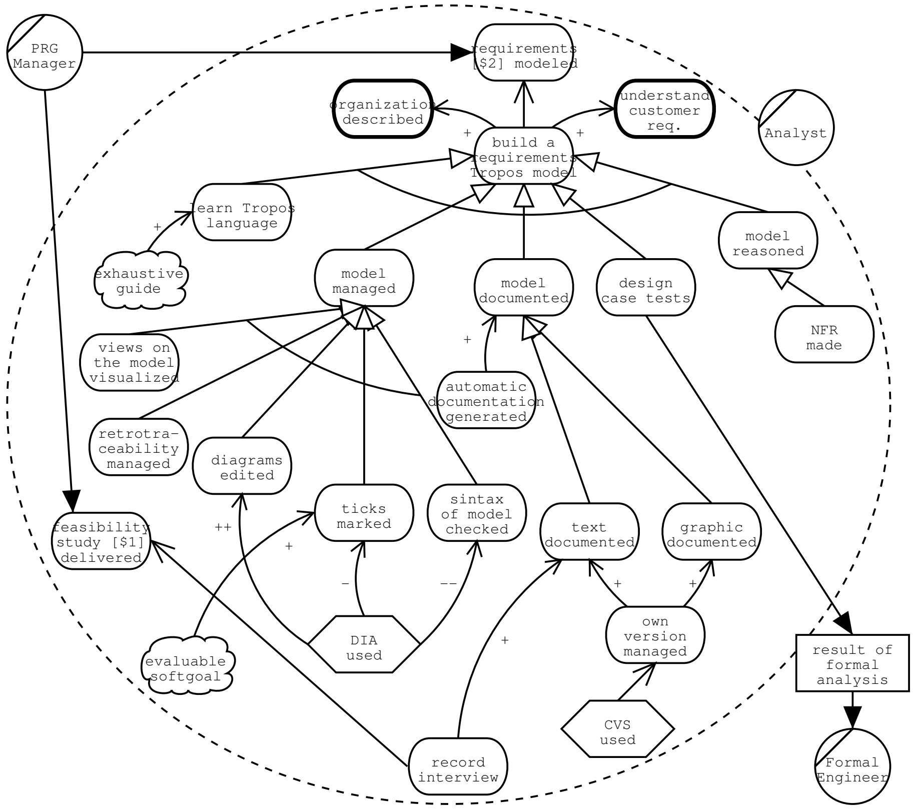

Fig. 4. Revising Goal analysis from the point of view of the `Analyst`.

namely `model documented`, has been OR decomposed into `text documented` and `graphic documented`. Finally, the goal `feasibility study [$1] delivered` (see Figure 1), 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 type 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.

Of course, during the goal analysis of the system-to-be 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 backward dependency on the social environment, unless for resources representing I/O flow)[5]. Also, the analysis of the system goals should be carried on within the scope of the system as far as possible, trying to minimize revisions of decisions already taken during early requirements analysis. Finally, the system-to-be is characterized only by delegated goals, which are, then, exclusively generated by the early requirement analysis in this simplified setting.

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 introduced here. In particular, it is relevant to show the very beginning of the late requirement analysis, and how a set of goals 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 this end, Figure 5 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 necessarily an unresolved social actor goal has to be resolved through a goal dependency on the system; instead, it may generate 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 5, it can be seen that `Analyst` depends on the `Tropos tool` for `NFR supported`, as well as the `Architecture Designer` does, although for fulfilling a different internal goal (`NFR evaluated` instead of `NFR made`). `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`, and, finally, on the softgoals `good user guide` and `good help on line`. As another example, let us 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 plan `NuSMV integrated`, with the motivations of `internal state visualized` and `counter-example provided`. Other examples, not commented here, are shown in Fig-

[5] Actually, some more dependencies could be introduced, especially if we consider the system in relations with other systems or its subsystems. This possibility, indeed, allow us for an interesting level of analysis for, e.g., the so called *autonomous systems*. In any case, the choice of relating the system with other systems, or decompose it into subsystems, comes later in the process, and falls out of the scope of the present paper.

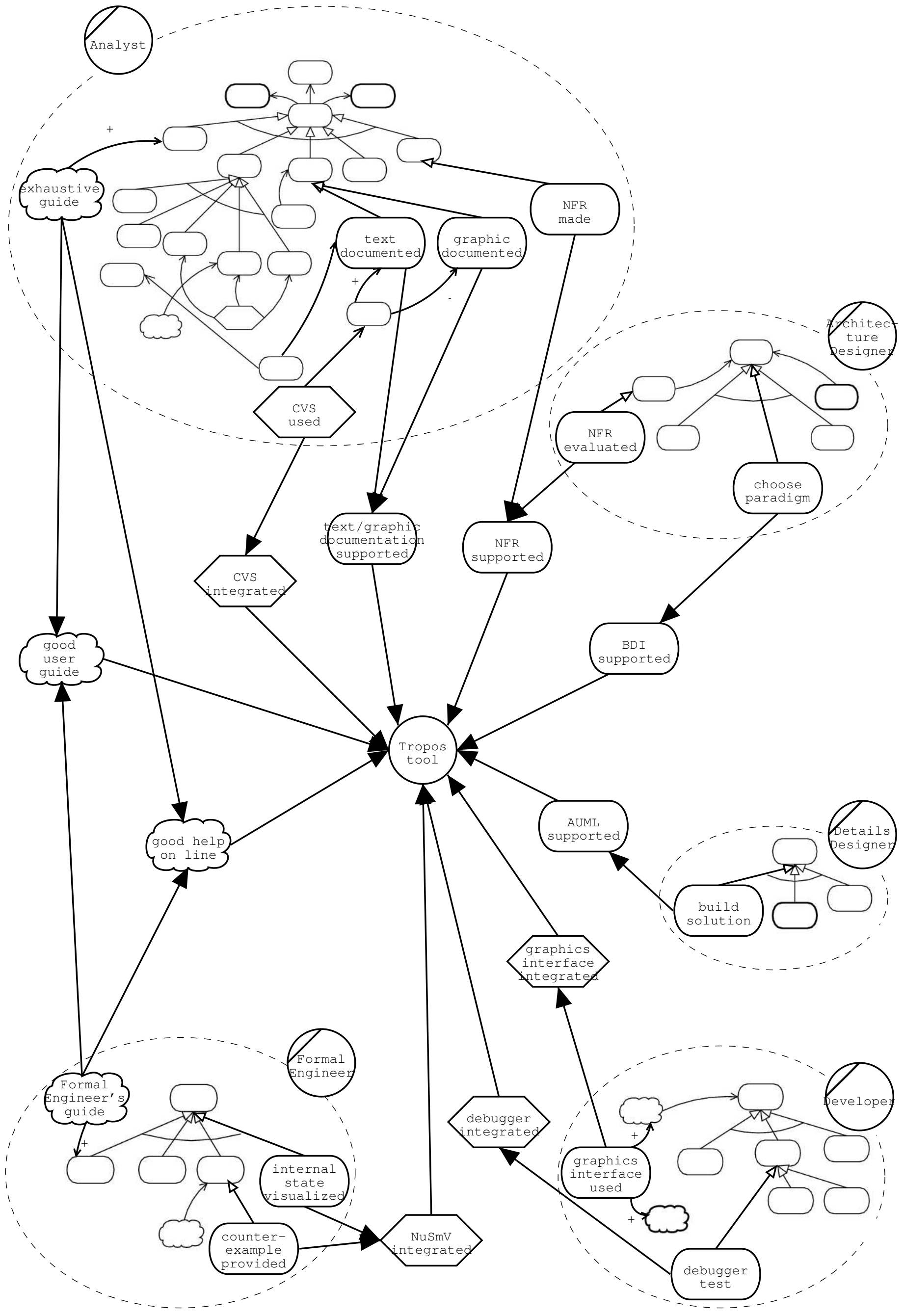

Fig. 5. Actor diagram: focus on the system actor `Tropos tool`.

ure 5, which, 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, which is currently 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, a stronger emphasis with respect to previous papers [11, 12, 18, 19, 4] 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 process of goals definition, analysis, and revision has been presented, pointing out, in particular, how the goal diagram construction has to be considered as an incremental process, based on a sequence of revision steps.

Finally, the last figure shown in the paper is aimed at giving an introductive hint on 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 developing 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 5, already provides for a first step towards the solution. Of course, the best support can be reached only through the development of an appropriate tool: the Tropos tool.

Acknowledgments

Our acknowledgments are addressed to all the IRST people working at the Tropos tool project. A special thanks is due to Eric Yu, who provided precious suggestions and comments during very preliminary discussions on the idea. We would also like to thank Angelo Susi and Paolo Busetta for comments on drafts of this paper.

References

1. B. Bauer, J. P. Müller, and J. Odell. Agent UML: A formalism for specifying multiagent software system. In P. Ciancarini and M. Wooldridge, editors, *Agent-Oriented Software Engineering – Proceedings of the First International Workshop (AOSE2000)*, volume 1957, pages 91–103, Limerick, Ireland, June 2000. Springer-Verlag Lecture Notes in Computer Science.
2. D. Bertolini, P. Bresciani, A. Daprà, A. Perini, and F. Sannicolò. Requirement Specification of a CASE tool supporting the Tropos methodology. Technical Report 0203-01, ITC-irst, via Sommarive, Povo, Trento, January 2002.
3. P. Bresciani, A. Perini, P. Giorgini, F. Giunchiglia, and J. Mylopoulos. Modelling early requirements in Tropos: a transformation based approach. In Wooldridge et al. [22], pages 151–168.

4. P. Bresciani and F. Sannicolò. Applying Tropos Requirements Analysis for defining a Tropos tool. In P. Giorgini, Y. Lespérance, G. Wagner, and E. Yu, editors, *Agent-Oriented Information System. Proceedings of AOIS-2002: Fourth International Bi-Conference Workshop*, pages 135–138, Toronto, Canada, May 2002.
5. P. Bresciani and F. Sannicolò. Applying Tropos to requirement analysis for a Tropos tool. Technical Report 0204-01, ITC-irst, via Sommarive, Povo, Trento, April 2002.
6. L. K. Chung, B. A. Nixon, E. Yu, and J. Mylopoulos. *Non-Functional Requirements in Software Engineering*. Kluwer Publishing, 2000.
7. 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):410–425, March 2000.
8. A. Dardenne, A. van Lamsweerde, and S. Fickas. Goal-directed requirements acquisition. *Science of Computer Programming*, 20(1–2):3–50, 1993.
9. P. Cederqvist et al. *Version Management with CVS*. http://www.cvshome.org/docs/manual/.
10. A. Fuxman, M. Pistore, J. Mylopoulos, and P. Traverso. Model Checking Early Requirements Specifications in Tropos. In *Proceedings Fifth IEEE International Symposium on Requirements Engineering (RE01)*, pages 174–181, Toronto, Canada, August 2001.
11. 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 2001.
12. F. Giunchiglia, A. Perini, and J. Mylopoulus. The Tropos Software Development Methodology: Processes, Models and Diagrams. In C. Castelfranchi and W.L. Johnson, editors, *Proceedings of the first international joint conference on autonomous agents and multiagent systems*, pages 63–74, palazzo Re Enzo, Bologna, Italy, July 2002. ACM press. Featuring: 6th International Conference on Autonomous Agents, 5th International Conference on MultiAgents System, and 9th International Workshop on Agent Theory, Architectures, and Languages.
13. F. Giunchiglia, A. Perini, and F. Sannicolò. Knowledge level software engineering. In J.-J.C. Meyer and M. Tambe, editors, *Intelligent Agents VIII*, LNCS 2333, pages 6–20, Seattle, WA, USA, August 2001. Springer-Verlag.
14. Gnome. *Dia Tutorial*. http://www.lysator.liu.se/∼alla/dia/diatut/all/all.html.
15. Knowledge Management Lab at the University of Toronto. *OME3 Documentation*. http://www.cs.toronto.edu/km/ome/docs/manual/manual.html.
16. J. Mylopoulos, L. Chung, S. Liao, H. Wang, and E. Yu. Exploring Alternatives during Requirements Analysis. *IEEE Software*, 18(1):92–96, February 2001.
17. J. Odell, H. V. D. Parunak, and B. Bauer. Extending UML for Agents. In G. Wagner, Y. Lesperance, and E. Yu, editors, *Proc. of Agent-Oriented Information System Workshop at the 17th National conference on Artificial Intelligence*, pages 3–17, Austin, TX, 2000.
18. A. Perini, P. Bresciani, P. Giorgini, F. Giunchiglia, and J. Mylopoulos. Towards an Agent Oriented approach to Software Engineering. In A. Omicini and M. Viroli, editors, *WOA 2001 – Dagli oggetti agli agenti: tendenze evolutive dei sistemi software*, Modena, Italy, 4–5 September 2001. Pitagora Editrice Bologna.
19. A. Perini, P. Bresciani, F. Giunchiglia, P. Giorgini, and J. Mylopoulos. A Knowledge Level Software Engineering Methodology for Agent Oriented Programming. In J. P. Müller, E. Andre, S. Sen, and C. Frasson, editors, *Proceedings of the Fifth International Conference on Autonomous Agents*, pages 648–655, Montreal CA, May 2001.
20. F. Sannicolò, A. Perini, and F. Giunchiglia. The Tropos modeling language. A User Guide. Technical Report 0204-13, ITC-irst, January 2002.
21. A. H. Watt. *Designing SVG web graphics*. D. Dwyer, 2002.

22. M.J. Wooldridge, G. Weiß, and P. Ciancarini, editors. *Agent-Oriented Software Engineering II*. LNCS 2222. Springer-Verlag, Montreal, Canada, Second International Workshop, AOSE2001 edition, May 2001.
23. 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.
24. E. Yu. *Modelling Strategic Relationships for Process Reengineering*. PhD thesis, University of Toronto, Department of Computer Science, University of Toronto, 1995.
25. E. Yu. Software Versus the World. In Wooldridge et al. [22], pages 206–225.
26. 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.