UNIVERSITY
OF TRENTO

DEPARTMENT OF INFORMATION AND COMMUNICATION TECHNOLOGY

38050 Povo – Trento (Italy), Via Sommarive 14
http://www.dit.unitn.it

APPLYING TROPOS METHODOLOGY TO A REAL CASE STUDY: COMPLEXITY AND CRITICALITY ANALYSIS

Maddalena Garzetti, Paolo Giorgini, John Mylopoulos,
and Fabrizio Sannicolò

October 2002

Technical Report # DIT-02-0097

Also: The paper has been accepted to the workshop "WOA 2002. Dagli OGGETTI agli AGENTI Dall'informazione alla Conoscenza"

Applying Tropos Methodology to a real case study: Complexity and Criticality Analysis

Maddalena Garzetti*, Paolo Giorgini*, John Mylopoulos†, and Fabrizio Sannicolò‡

*Department of Information and Communication Technology, University of Trento,
Trento, Italy, Email: {maddalena.garzetti,paolo.giorgini}@dit.unitn.it

†Department of Computer Science, University of Toronto,
Toronto, Canada, Email: jm@cs.toronto.edu

‡ITC-irst, via Sommarive 18, 38055 Povo, Trento, Italy,
Email: sannico@irst.itc.it

***Abstract*— Currently in Requirements Engineering the attention is being focused more and more on the understanding of a problem by studying the existing organizational setting in which the system will operate. In this paper we present the application of the Tropos early requirements analysis to a real case study, the *Ice Co*. We introduce a new type of analysis for actor diagrams based on two different parameters, *complexity* and *criticality*, and we show the results we obtained during the case study.**

I. Introduction

The development of a successful software system and in particular of a multi-agent system, relies on the understanding of the organizational context within which the system will operate and how the system will be part of the encompassing organizational processes. For this reason, in Requirements Engineering the attention is being focused more and more on the very early phase of software development in which the system is studied with its context as a larger social-technical system.

According to this perspective, we are developing *Tropos* [2], [16], [11], [9], [10], an agent-oriented software engineering methodology characterized by three keys aspects [15], [5], [4], [5]. First, it uses concepts like *actors*, *goals*, *softgoals*, *plans*, *resources*, and *intentional dependencies* along all the phases of software development. Second, it pays great attention to the activities that precede the specification of the perspective requirements, like understanding *how* and *why* the intended system would meet the organizational goals[1]. 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].

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. In particular, a dependency relates one *depender*, one *dependee*, and one *dependum*. The depender and the dependee are actors, while the dependum may be goals, softgoals, plans, and resource. The depender delegates the fulfilling of own dependum to another actor, the dependee.

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's global architecture in terms of subsystems, represented as actors, and their data dependencies, represented as actor dependencies.

The **detailed design** phase aims at specifying each architectural component in further detail (adopting a subset of the AUML diagrams [14], [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.

The present paper focuses on the Tropos early requirements analysis phase applied to a real case study, the *Ice Co*. The case study is part of a jointly run project involving University of Trento, Istituto Trentino di Cultura (ITC-irst) and Centro Ricerche Fiat (CRF). The objective of the project is to build a system for facilitating access by industries in Trentino to the logistic services available on the web. Ice Co is one of the companies that we have interviewed and analyzed in order to understand their needs and then define important and concrete requirements of the system we want to develop. Three full time people worked for four months interviewing people inside Ice

[1] In particular, Tropos is widely inspired by Eric Yu's framework for requirements engineering, called i^*, which offers actors, goals, and actors dependencies as primitive concepts [19], [20], [21].

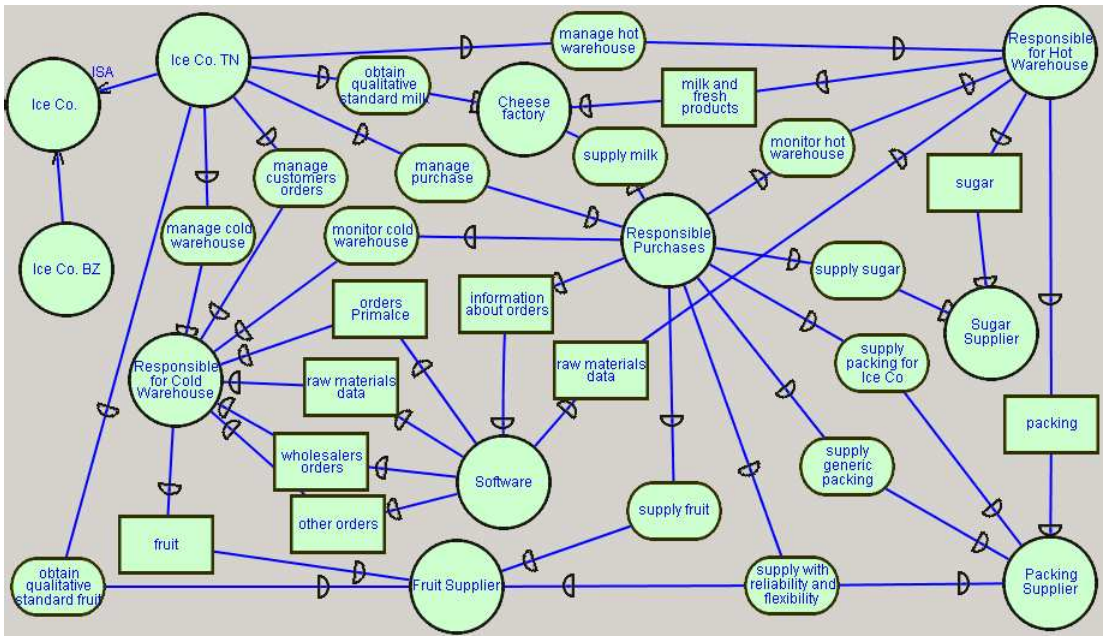

Fig. 1. The actor diagram for the Ice Co case study with respect to the in-bound problem.

Co and refining with them the Tropos models. Currently, the project has started the second phase of the requirements analysis, the Tropos late requirements analysis, that should be concluded by the end of the year.

The organizational environment of Ice Co has been modeled by means of networks of social dependency relationships among actors (actor diagrams) and the impact of each relationship over the organization has been evaluated using a new type of analysis based on two different parameters: *complexity* and the *criticality*. The analysis helped us to discover inside the organization some imbalance between the actors in terms of complexity of the goals assigned to them and criticality of their responsibilities with respect to the overall organization. This information is used to motivate and define the functional requirements of the system that will be introduced in the organization.

The rest of this paper is structured as follows. Section II presents briefly the case study and Section III introduces a portion of the Tropos early requirements analysis for it. Section IV presents the complexity and criticality analysis and the results obtained for the case study. Section V compares Tropos with other relevant methodologies. Conclusion and directions for further research are presented in Section VI.

II. The Ice Co case study

The Ice Co is an Italian company that produces and trades in ice-cream and frozen confectionery in the region of Trentino/Alto-Adige (Italy). Ice Co has two different factories, one in the province of Trento where they produce ice-cream, and another one in the province of Bolzano where they produce frozen confectionery. In each factory there is a warehouse, where raw materials such as milk, fruit, and packages are stored, a plant where ice cream or frozen confectionery is produced, and another warehouse where the end products are stocked. Moreover, Ice Co owns part of PrimaIce, a service company which distributes end products to the Ice Co's customers situated both in Trentino/Alto-Adige and throughout the rest of Italy. PrimaIce is responsible for goods between the warehouse situated in Trentino and that situated in Alto Adige.

All Ice Co activities can be referred to three main problems:

- In-bound. It consists of two sub-problems: *(i)* the selection, among several candidates, of one or more suppliers for each sort of raw material and *(ii)* how to store the different kinds of raw materials inside the warehouses (e.g., milk and fruit have to be kept at the temperature of $-27°$C).
- Production. It concerns the internal organization of Ice Co. In particular, it addresses the production problem that includes decisions like, for instance, the activation and the shutting-down of a production line, including who take such decisions and under what conditions.
- Out-bound. It concerns the distribution of the end products. This includes problems related to the specific customers, such as wholesalers, bars, and restaurants.

In this paper we use and describe only a portion of the Ice Co case study, and in particular we consider the in-bound and out-bound problems. More details about the case study are available in [8].

III. Early Requirements Analysis

In early requirement analysis, we model and analyze the intentions of the stakeholders. These are modeled as goals that, through some form of analysis [13], [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. We use *actor diagrams* for describing the network of social dependency relationships among actors and *goal diagrams* for analyzing goals. However, in this paper we present only the analysis of the actor diagrams for the Ice Co and we omit for lack of space goal analysis.

Graphically in an actor diagram, actors are depicted as circles, their goals as ovals, and a dependency relationships between two actors (the depender and the dependee) as two arrowed lines connected by a graphical symbol varying according to the dependum: a goal (oval), a plan (hexagon) or a resource (rectangle). In the actor diagram depicted in Figure 1, Ice Co is modeled as the actor `Ice Co.` and it is specialized (*IsA*) in `Ice Co. TN` and `Ice Co. BZ` (i.e., the factories based in Trentino and in Alto Adige, respectively).

`Ice Co. TN` depends on the actor `Responsible for Cold Warehouse` (responsible for the warehouse containing raw materials like fruit) for fulfilling the goals `manage cold warehouse` and `manage customers orders`. Moreover, it depends on the `Responsible for Hot Warehouse` for `manage hot warehouse` and on the `Responsible Purchases` for the goal `manage purchase`.

`Responsible Purchases` has the main goal to buy raw materials, like sugar, milk, and packing, when it is needed, and it relies on `Responsible for Cold Warehouse` and `Responsible for Hot Warehouse` for monitoring the warehouses (respectively, `monitor cold warehouse` and `monitor hot warehouse`). `Responsible Purchases` purchases raw materials from the following suppliers: `Cheese Factory` (dependum `supply milk`), `Sugar Supplier` (dependum `supply sugar`), `Fruit Supplier` (dependum `supply fruit`), and `Packing Supplier` —in this last case, we distinguish between packing for ICe Co products and generic packing— . Suppliers deliver raw materials to the warehouses; for instance, the `Responsible for Cold Warehouse` depends on the `Fruit Supplier` for the resource `fruit`, and analogously, `Responsible for Hot Warehouse` depends on the `Cheese Factory` for obtaining `milk and fresh products` and on `Packing Supplier` for obtaining the resource `packing`. Finally, Ice Co uses a software system (`Software`) to support its activities, such as for instance the customers' orders management and the requests of raw materials to the suppliers. `Software` depends on `Responsible for Cold Warehouse` for the resources `raw materials data`, `orders PrimaIce`, `wholesalers data`, and `other orders`, and on `Responsible for Hot Warehouse` for `raw materials data`. On the other hand, `Software` has to provide the information about orders to the actor `Responsible Purchases`.

Figure 2 shows the actor diagram about the out-bound problem and in particular, it shows the dependencies between the actors of the Alto Adige factory involved in goods exchanging (we have a similar diagram for the Trentino factory). `Responsible Purchases BZ` (`BZ` stands for Bolzano) is delegated for managing purchases and `Responsible for Cold Warehouse BZ` for managing warehouse in Bolzano. As mentioned in Section II, although some goods are produced and packed in the province of Trento, and others in the province of Bolzano, Ice Co distributes its products in the whole regional territory. So, in order to exchange goods between Bolzano and Trento, the `Responsible Purchases BZ` orders directly to the `Responsible for Cold Warehouse` (goal dependency `order goods TN`) and analogously, the responsible purchase of Trento orders goods to the `Responsible Cold Warehouse BZ`.

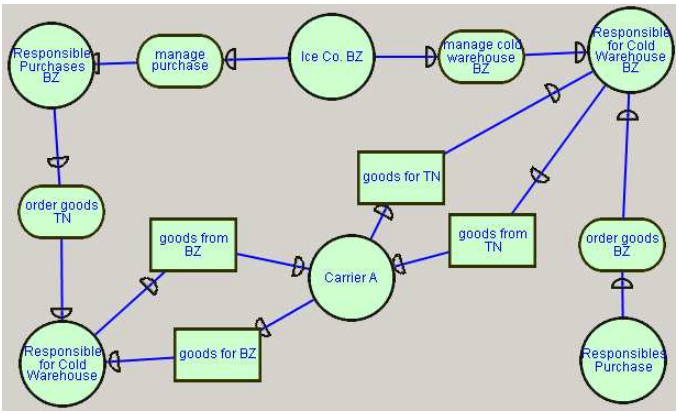

Fig. 2. Actor diagram modeling the out-bound problem.

`Carrier A` is delegated by PrimaIce for transporting goods between Trento and Bolzano. It receives from `Responsible Cold Warehouse BZ` goods to be transferred to Trento (`goods for TN`) and provides the `Responsible Cold Warehouse BZ` with goods from Trento (`goods for TN`). Analogously, `Carrier A` transfers goods from Bolzano to Trento. Due to lack of space, the other actor diagrams are not presented here, but can be found in [8].

IV. Complexity and Criticality Analysis

In the previous section we have partially analyzed the Ice Co case study by actors diagrams. Although Tropos allows us to refine the model by means of the transformational approach [3] and others techniques of analysis and decomposition [13], [6], [17], it is worth, from our point of view, reasoning about other aspects nor addressed up to now. In particular, during the interviews inside Ice Co, people expressed the requirement to differentiate each dependency with respect to the relevance and importance that the dependency assumes for the organization. For example, they manifested the necessity to express the importance of receiving fruit in time from a supplier and to manage effectively customers' orders. Moreover, according to them, a dependency should be characterized by a degree of complexity

Local scores – Complexity and criticality					
	Depender	*Dependum*	*Dependee*	*Complexity*	*Criticality*
1	Ice Co. TN	manage cold warehouse	Responsible for Cold Warehouse	3	3
2	Ice Co. TN	manage customers orders	Responsible for Cold Warehouse	2	3
3	Ice Co. TN	manage purchase	Responsible Purchases	3	2
4	Ice Co. TN	manage hot warehouse	Responsible for Hot Warehouse	2	2
5	Responsible for Hot Warehouse	sugar	Sugar Supplier	1	1
6	Responsible for Hot Warehouse	milk and fresh products	Cheese Factory	1	3
7	Responsible for Hot Warehouse	packing	Packing Supplier	1	2
8	Responsible for Cold Warehouse	fruit	Fruit Supplier	1	3
9	Responsible Purchases	monitor hot warehouse	Responsible for Hot Warehouse	1	2
10	Responsible Purchases	manage cold warehouse	Responsible for Cold Warehouse	3	3
11	Responsible Purchases	supply milk	Cheese Factory	1	3
12	Responsible Purchases	supply fruit	Fruit Supplier	3	3
13	Responsible Purchases	supply sugar	Sugar Supplier	1	1
14	Responsible Purchases	information about orders	Software	1	1
15	Responsible Purchases	supply generic packing	Packing Supplier	2	2
16	Responsible Purchases	supply packing for Ice Co.	Packing Supplier	1	2
17	Ice Co. TN	obtain qualitative standard milk	Cheese Factory	3	3
18	Ice Co. TN	obtain qualitative standard fruit	Fruit Supplier	2	3
19	Responsible Purchases	supply with reliability and flexibility	Packing Supplier	2	3
20	Responsible Purchases	supply with reliability and flexibility	Fruit Supplier	3	3
21	Software	orders PrimaICe	Responsible for Cold Warehouse	1	3
22	Software	raw materials data	Responsible for Cold Warehouse	1	3
23	Software	other orders	Responsible for Cold Warehouse	1	3
24	Software	wholesalers orders	Responsible for Cold Warehouse	1	3
25	Software	raw materials data	Responsible for Hot Warehouse	1	3

TABLE I

ASSIGNMENT VALUES TO EACH DEPENDENCIES REPRESENTED IN FIGURE 1.

representing the amount of the effort and resources needed for its achievement. So for example, managing the cold warehouse is a goal that requires more effort than that of managing the hot warehouse.

We propose in the following an analysis of the dependencies based on two different kinds of measures:

> ***complexity*** *is the measure of the effort required from the depender for achieving the dependum*,
>
> ***criticality***[2] *is the measure of how the goals of an actor will be affected if the dependum is not achieved.*

Complexity allows us to evaluate the amount of the effort that is required from each actor for achieving its responsibilities, and to analyze the whole actor diagram to discover possible overloads on some actors with respect to the others. Analogously, it is possible to use criticality to evaluate the criticality of an actor for the rest of the organization. We distinguish between *ingoing* and *outgoing* criticality. Ingoing criticality represents the criticality that an actor assumes when it is responsible for achieving a dependum, namely when the actor assume the role of dependee in the dependency. The outgoing critical-

[2]This definition is an extension of that one introduced by Eric Yu in his PhD thesis [20].

ity represents the criticality of the achievement of a dependency for the depender. Basically, given a dependency we assign to it a value of criticality that assume a different meaning for the depender (outgoing criticality) and the dependee (ingoing criticality).

Table I reports the values of complexity and criticality assigned to each dependency of the actor diagram in Figure 1. In our analysis, criticality and complexity can assume values 1 (*low*), 2 (*medium*) and 3 (*high*), but of course the range of values can be extended.

For each actor of the actor diagram we calculate the *global complexity* as the sum of the complexity of the dependencies where the actor is the dependee. The *global ingoing criticality* and *global outgoing criticality* are the sum of the criticality of the dependencies where the actor is the dependee and depender, respectively. These global values allow us to evaluate the overall organization in terms of a distribution of complexity and criticality. So for instance, we can discover that there are some actors that are particularly critical for the overall organization's objectives or that other actors are overloaded in terms of complexity in their responsibilities. This kind of evaluations are particularly useful for discovering important requirements of a software system that eventually will be implemented and adopted inside the organization, but of course, such information can be also useful to redistribute the responsibilities and the activities among the actors. In this work we use the global values to define the functional requirements of a software system that will be adopted to support the organization.

Figure 3 shows the procedure we adopt for the complexity and criticality analysis. Basically, in the first part of the procedure we calculate for each actor the global values for complexity and criticality (ingoing and outgoing) and then we build two lists *comp-actorList* and *crit-actorList* in which we insert all the actors for which the global values of complexity and criticality, respectively, are greater than the max values they can assume, namely *actor.max-complexity* and *actor.max-criticality*. We suppose that it is possible to define for each actor a max value of complexity and criticality based on the actor's competences, abilities and role it assumes inside the organization. Finally, the procedure ends with the assignment of some dependencies in which the actors, contained in the *comp-actorList* and *crit-actorList*, are included in the software system we want to develop. The idea is that first (*assign-comp(comp-actorList)*) we assign to the system goals in order to reduce the complexity of the actors in *comp-actorList* and then (*assign-crit(crit-actorList)*) to reduce criticality of actors in *crit-actorList*.

Table II shows the global complexity and critically values for the actor diagrams presented in Figure 1 and Figure 2. *N.o.D.* is the the number of dependencies in which each actor is involved and for which global values are calculated.

Figure 4 presents the revised actor diagram in which we have introduced two new actors (software systems) `S1` and `S2`. Assigning to `S1` the responsibility of processing and updating the information used by software system (`Software`) we can reduce the value of the ingoing criticality of the `Responsible for Cold Warehouse`, that decreases from 24 to 9. Similarly, the introduction of `S2` allows us to reduce from 3 to 2 the ingoing criticality of the `Responsible Purchases`.

```
global actorList, dependencyList, comp − actorList,
       crit − actorList;
procedure weightDependency(actorList, dependencyList,
          comp − actorList, crit − actorList)
  begin
    comp − actorList := crit − actorList := nil;
    for actor in actorList
      actor.complexity := 0
      actor.criticality − in := actor.criticality − out := 0;
      for dependency in dependencyList
        if dependency.depender = actor then
        actor.criticality − out := actor.criticality − out+
                                   dependency.criticality;
        if dependency.dependee = actor then
        begin
          actor.complexity := actor.complexity+
                              dependency.complexity;
          actor.criticality − in := actor.criticality − in+
                                    dependency.criticality;
        end ;
      end ;
      if actor.complexity > actor.max − complexity then
      add(actor, comp − actorList);
      if actor.criticality − in > actor.max − criticality then
      add(actor, crit − actorList);
    end ;
  end ;
  assign − comp(comp − actorList);
  assign − crit(crit − actorList);
end procedure
```

Fig. 3. The procedure for the complexity and criticality analysis.

V. Related work

As reported in [2], [16], [11], [9], [10], one of most topic feature of the Tropos methodology is that it aspires to span the overall software development process. This fact is depicted in Figure 5 which shows the relative coverage of Tropos as well as i^* [20], KAOS [6], GAIA [18], AAII [12], MaSE [7], and AUML [14], [1]. Tropos covers the software development process as a whole, from the early steps, in which the software engineer picks up and models requirements of the organizational setting (early requirements) and of the system-to-be (late requirements), up to the detailed design where the design is carried out by means, for example, of a series of AUML activity diagrams (for more details about the AUML diagrams in Tropos, see [2]). Moreover, such methodology uses the same concepts, like, for example, actor, goal, softgoal, and goal dependencies,

Global scores – Complexity, Ingoing, Outgoing criticality							
	Actor	*Complexity*	*N.o.D.*	*Ingoing criticality*	*N.o.D.*	*Outgoing criticality*	*N.o.D.*
1	Ice Co. TN	0	0	0	0	16	6
2	Responsible Purchases	3	1	3	1	27	11
3	Responsible for Cold Warehouse	15	8	24	9	4	2
4	Responsible for Hot Warehouse	4	3	7	3	6	3
5	Cheese Factory	5	3	9	3	0	0
6	Sugar Supplier	2	2	2	2	0	0
7	Fruit Supplier	8	4	12	4	0	0
8	Packing Supplier	5	4	9	4	0	0
9	Software	1	1	3	1	15	5
10	Career A	2	2	2	2	2	2
11	Responsible Purchases BZ	3	1	3	1	2	1
12	Ice Co. BZ	0	0	0	0	6	2
13	Responsible for Cold Warehouse BZ	6	3	6	3	1	1

TABLE II

TABLE REPRESENTING ALL THE VALUES FOR INGOING AND OUTGOING DEPENDENCIES OF EACH ACTOR.

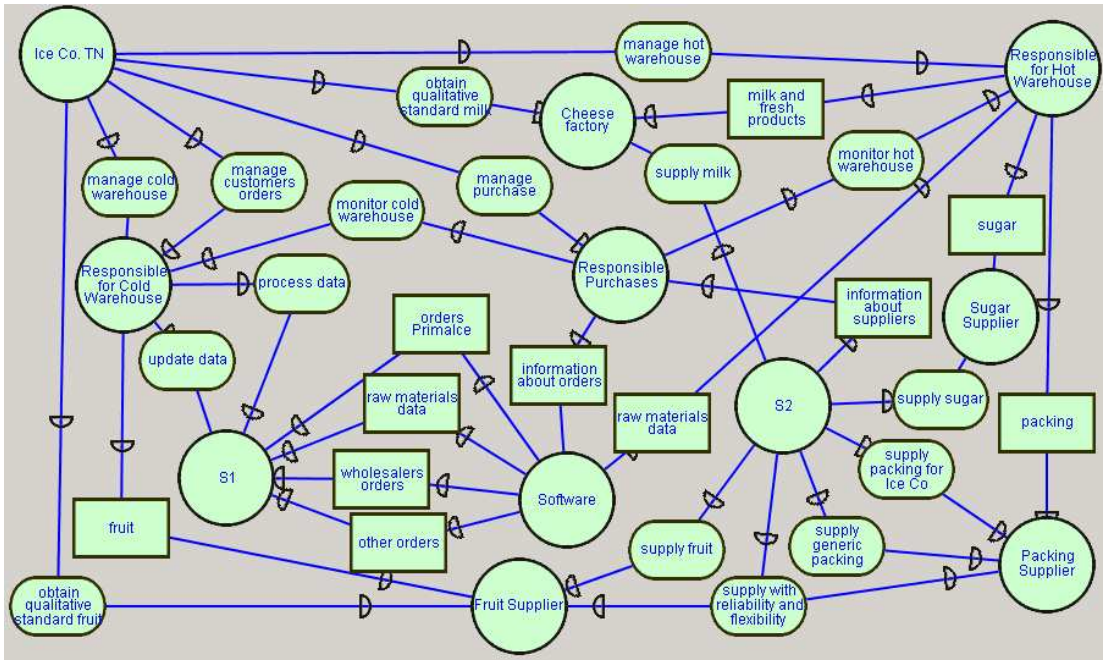

Fig. 4. Actor diagram modeling the in-bound problem after the revising.

along all the phases; this feature is not addressed from other methodologies.

VI. Conclusion and Future Work

In this paper we have presented the Tropos early requirement analysis for a real case study, the Ice Co, and we have introduced a new type of analysis for actor diagrams based on two different parameters: *complexity* and *criticality*. The analysis helped us to discover inside the organization some imbalance between the actors in terms of complexity of the goals assigned to them and criticality of their responsibilities with respect to the overall organization. This information is used to motivate and define the functional requirements of the system that will be introduced in the organization.

The case study has been analyzed within a running project in which we are currently involved. The use of the Tropos methodology has been extremely positive and in particular it has been an effective means of interaction between us and the people interviewed. It allowed us to show after the interviews our under-

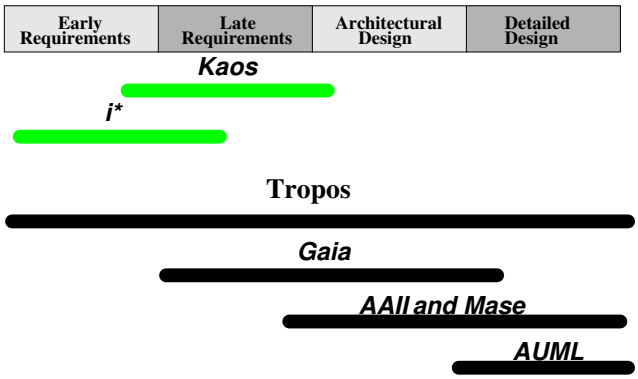

Fig. 5. Comparison of Tropos with other software development methodologies.

standing of the problem and to discuss with our customers the possible requirements to consider in the final system. Differently form other modeling languages, like for instance UML, the Tropos graphical notation and the concepts used in it are more intuitive and comprehensible for people that are not expert in software engineering.

We are currently applying the Tropos late requirements analysis to the second phase of our project and we are defining new type of analysis for it. Moreover, we are working to an efficient algorithm in order to automate the process of reassignment of the dependencies in the actor diagrams.

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] P. Bresciani, P. Giorgini, F. Giunchiglia, J. Mylopoulos, and A. Perini. TROPOS: An Agent-Oriented Software Development Methodology. Submitted to the Journal of Autonomous Agents and Multi-Agent Systems. Kluwer Academic Publishers, March 2002.

[3] P. Bresciani, A. Perini, P. Giorgini, F. Giunchiglia, and J. Mylopoulos. Modelling early requirements in Tropos: a transformation based approach. In M.J. Wooldridge, G. Weiß, and P. Ciancarini, editors, *Agent-Oriented Software Engineering II*, LNCS 2222, pages 151–168. Springer-Verlag, Montreal, Canada, Second International Workshop, AOSE2001 edition, May 2001.

[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. Requirements Analysis in Tropos: a self referencing example. In *Net.ObjectDays 2002 - Workshop on Agent Technology and Software Engineering (AgeS02)*, pages 100–114, October 2002.

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

[7] S. A. Deloach. Analysis and Design using MaSE and agentTool. In *12th Midwest Artificial Intelligence and Cognitive Science Conference (MAICS 2001)*, Miami University, Oxford, Ohio, March 31 - April 1 2001.

[8] M. Garzetti, P. Giorgini, and F. Sannicolò. The Ice Co case study: Requirements Analysis. Technical report, University of Trento, via Sommarive 18, Povo, I-38050, Trento-Povo, July 2002. URL - http://neven.science.unitn.it/∼logicost/documents/IceCo.doc.

[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)*, pages 283–290, Buenos Aires - ARGENTINA, June 2001.

[10] 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.

[11] 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.

[12] D. Kinny, M. Georgeff, and A. Rao. A Methodology and Modelling Technique for Systems of BDI Agents. In W. Van de Velde and J. W. Perram, editors, *Agents Breaking Away: Proceedings of the Seventh European Workshop on Modelling Autonomous Agents in a Multi-Agent World*, Springer-Verlag: Berlin, Germany, 1996.

[13] J. Mylopoulos, L. Chung, S. Liao, H. Wang, and E. Yu. Exploring Alternatives during Requirements Analysis. *IEEE Software*, 18(1):92–96, February 2001.

[14] 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.

[15] 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.

[16] 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.

[17] F. Sannicolò, A. Perini, and F. Giunchiglia. The Tropos modeling language. A User Guide. Technical Report 0204-13, ITC-irst, January 2002.

[18] M. Wooldridge, N. R. Jennings, and D. Kinny. The Gaia Methodology for Agent-Oriented Analysis and Design. *Autonomous Agents and Multi-Agent Systems*, 3(3):285–312, 2000.

[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.

[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 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.