Supporting Requirements Analysis in Tropos: a Planning-Based Approach

Volha Bryl[1], Paolo Giorgini[1], John Mylopoulos[1,2]

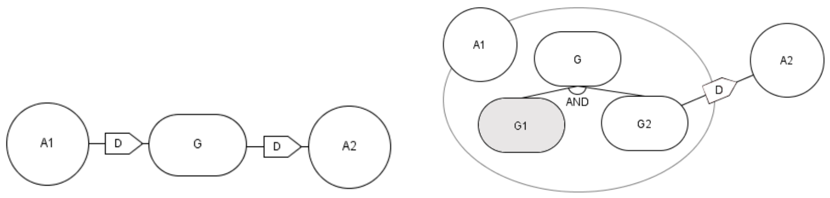
[1] University of Trento, Italy,
[2] University of Toronto, Canada
bryl@dit.unitn.it, paolo.giorgini@unitn.it, jm@cs.toronto.edu

Abstract. Software systems are becoming more and more part of human life influencing organizational and social activities. This introduces the need of considering the design of a software system as an integral part of the organizational and social structure development. Alternative requirements and design models have to be evaluated and selected from a social perspective finding a right trade-off between the technical and social dimension. In this paper, we present a Tropos-based approach for requirements analysis, which adopts planning techniques for exploring the space of requirements alternatives and a number of social criteria for their evaluation. We describe the tool-supported analysis process with the help of a case study (the e-voting system), which is a part of a project funded by the Autonomous Province of Trento.

1 Introduction

Unlike their traditional computer-based cousins, socio-technical systems include in their architecture and operation organizational and human actors along with software ones. Moreover, human, organizational and software actors in such systems rely heavily on rich inter-dependencies to other actors in order to fulfill their respective objectives. Not surprisingly, an important element in the design of socio-technical systems is the identification of a set of dependencies among actors which, if respected by all parties, will fulfill all objectives.

Let's make the problem more concrete. KAOS [5] is a state-of-the-art requirements elicitation technique that starts with stakeholder goals and through a systematic, tool-supported process derives functional requirements for the system-to-be and a set of assignments of leaf-level goals (constraints, in KAOS terminology) to external actors so that if the system-to-be can deliver the functionality it has been assigned and external actors deliver on their respective obligations, stakeholder goals are fulfilled. However, there are (combinatorially) many alternative assignments to external actors and the system-to-be. How does the designer choose among these? How can we select an optimal, or "good enough" assignment? What is an optimal assignment? The KAOS framework remains silent on such questions. Alternatively, consider the $i*$ and Tropos frameworks for modeling and analysis of early requirements of agent-oriented systems [11, 1]. In $i*$/Tropos, goals are explicitly associated with external stakeholders and they



(a) 1st alternative (b) 2nd alternative

Fig. 1. Sample problem: two alternative models

can be delegated to other actors or the system-to-be. Or, they can be decomposed into subgoals that are delegated to other actors. In this setting, selecting a set of assignments is more complex because delegations can be transitive and iterative. “Transitive” means that actor A may delegate goal G to actor B who in turn delegates it to actor C. “Iterative” means that an actor A who has been delegated goal G, may choose to decompose it (in terms of an AND/OR decomposition) and delegate its subgoals to other actors.

To illustrate the problem, consider the design task in Figure 1 where actor A_1 has to achieve goal G, which can be refined into two subgoals G_1 and G_2. The actor can decide to achieve the goal by itself or delegate it to actor A_2. In both cases, there are a number of alternative ways that can be adopted. For instance, A_1 can decide to delegate to A_2 the whole G (Figure 1a), or a part of it (Figure 1b). The diagrams follow Tropos modelling notation with circles representing actors, big dashed circles representing actors’ perspective, and ovals representing goals (interconnected by AND/OR-decomposition links). Social dependencies among actors for goals are represented by D-labelled directed links. Even for such a simple example, the total number of alternative requirements models is big, and a systemized approach and tool support for constructing and evaluating such networks of delegations would be beneficial.

We are interested in supporting the design of socio-technical systems, specifically the design of a set of inter-actor dependencies intended to fulfill a set of initial goals. The support comes in the form of a tool that is founded on an off-the-shelf AI planner to generate and evaluate alternative assignments of actor dependencies to identify an optimal design.

Specifically, our tool solves the following problem: given a set of actors, goals, capabilities, and social dependencies, the tool generates alternative actor dependency networks on the basis of the following steps, which may be interleaved or used inside one another:

– Check problem-at-hand: (a) Analyze actor capabilities: check whether existing actors possess enough capabilities to collectively satisfy their goals; (b) Analyze actor inter-dependencies: check whether existing dependencies between actors allows them to fulfill all given goals.

– Explore alternative dependency networks: (a) With the help of planning algorithms construct assignments of goals to actors that leads to the satisfaction of the actors' goals; (b) Evaluate alternatives by assessing and comparing them with respect to a number of criteria, provided by the designer.

The idea of casting the problem of designing a set of delegations for a socio-technical system into a planning one has already been presented in [4, 3]. In addition, [3] proposes a preliminary approach to the evaluation of alternatives. However, what was missing in these works, is the *systematic process* of requirements analysis to support the design of socio-technical systems. In this paper, we present such a tool-supported process, which combines formalization and analysis of an organizational setting, the use of planning techniques [3], and a concrete set of optimality criteria to evaluate the produced requirements models. The approach is illustrated with the help of a case study involving an e-voting system, developed for the Autonomous Province of Trento. We also report on preliminary experimental results aimed to evaluate the scalability of the prototype tool.

The rest of the paper is structured as follows. Sections 2 and 3 present the baseline for this research [3, 4], namely, in Section 2 the details on applying planning techniques to requirements models construction are given, and in Section 3 criteria and procedures for evaluating those models are discussed. Section 4 presents the general process schema of the proposed requirements analysis approach. This is followed by the explanations on how the inputs to the process are organized and analyzed in Section 5. Section 6 introduces the e-voting case study and illustrates the whole approach on its basis, which is followed by summarizing remarks and discussion on the future work directions in Section 7.

2 Using Planning to Construct Requirements Models

The task of constructing delegation networks can be framed as a *planning problem*: selecting a suitable design corresponds to selecting a plan that satisfies actors' and organizational goals [3]. In general, AI (Artificial Intelligence) planning [10] is about automatically determining the course of actions (i.e. a plan) needed to achieve a certain goal where an action is a transition rule from one state of the world to another. A specification language is required to represent the planning domain, i.e. the initial and the desired states and the actions.

The following predicates are used to formally represent the initial and the desired states of the world (i.e. the organizational setting). The predicates take variables of three types: `actors`, `goals` and `goal types`.

To typify goals, `type(goal, g_type)` predicate is used. Actor capabilities are described with `can_satisfy(actor, goal)` and `can_satisfy_gt(actor,g_type)` predicates meaning that an actor has enough capabilities to satisfy either a specific goal, or a goal of a specific type. Dependencies among actors are reflected by `can_depend_on(actor, actor)` and `can_depend_on_gt(actor, actor, g_type)` predicates, which mean that one actor can delegate to another actor the fulfilment of any goal or, in the latter case, any goal of a specific type. Predefined ways of goal refinement are represented by `and/or_decomposition(goal, goal, ..)`

```
A1, A2 − t_actor
G, G1, G2 − t_goal
(can_satisfy A1 G)
(can_satisfy A1 G1)
(can_satisfy A1 G2)
(can_satisfy A2 G)
(can_satisfy A2 G1)
(can_satisfy A2 G2)
(can_depend_on A1 A2)
(and_decomposition2 G G1 G2)
(wants A1 G)
```

(a) Formalization

```
DELEGATES   A1 A2 G
SATISFIES   A2 G
```

(b) Plan for 1st alternative

```
AND_DECOMPOSES   A1 G G1 G2
SATISFIES   A1 G1
DELEGATES   A1 A2 G2
SATISFIES   A2 G2
```

(c) Plan for 2nd alternative

Fig. 2. Sample problem: formalization and plans.

predicates. Initial actor desires are represented with `wants(actor, goal)` predicate. When a goal is fulfilled `satisfied(goal)` predicate becomes true for it. In Figure 2a the example presented in Figure 1 (hereafter referred to as *sample problem*) is formalized using the above described predicates.

A plan, which is constructed to fulfill actors' and organizational goals, comprises the following actions: (a) *Goal satisfaction*, (b) *Goal delegation*, and (c) *Goal decomposition*. Actions are described in terms of preconditions and effects, both being conjunctions of the above mentioned predicates and/or their negations. If a precondition of an action is true in the current state, then the action is performed; as a consequence of an action, a new state is reached where the effect of the action is true.

When the problem domain and the problem itself are formally represented, a *planner* is used to produce a solution. Having analyzed a number of available off-the-shelf planners (see [4] for the details), we have chosen LPG-td [7], a fully automated system for solving planning problems, which supports PDDL 2.2 specification [6], for implementing our planning domain. In Figure 2b-c two plans for the sample problem generated by LPG-td are shown.

A set of experiments, which we do not report here due to the space limits, were conducted with LPG-td to assess the scalability of the approach. The key point of the experiments was to identify the complexity limits (in terms of number of goals, actors and actor dependencies, depth of goal decomposition trees, etc.) which the planner could handle. The obtained results justify the use of planning in the requirements engineering domain as, according to our experience, requirements models of real life case studies stay within the complexity limits which our planning approach is able to manage. In [3] we have presented P-Tool, an implemented prototype to support the designer in the process of exploring and evaluating alternatives. The tool has the interface for the input of actors, goals and their properties. LPG-td is built in the tool, and is used to generate alternative requirements structures, which are then represented graphically using Tropos notation.

3 Evaluating Requirements Models

After a requirements model has been constructed, it can be evaluated from the two perspectives, global (i.e. the perspective of the designer), and local (i.e. the perspective of an individual actor). The perspective of the designer will normally include a number of non-functional requirements, which he wants to be met. E.g., these might be security concerns, efficiency, maintainability, user friendliness, etc. Evaluation is either quantitative, or qualitative, depending on whether one can "measure" the requirement in terms of numbers, or there exists just a relative scale to compare two models. An example of quantitative criteria would be a "fair", or homogeneous distribution of workload among the actors, with a variance as a measure of it. An example of qualitative criteria could be user friendliness, meaning that the expertise of the designer allows him to say that organizing the process in this way is "more user friendly" than in this other.

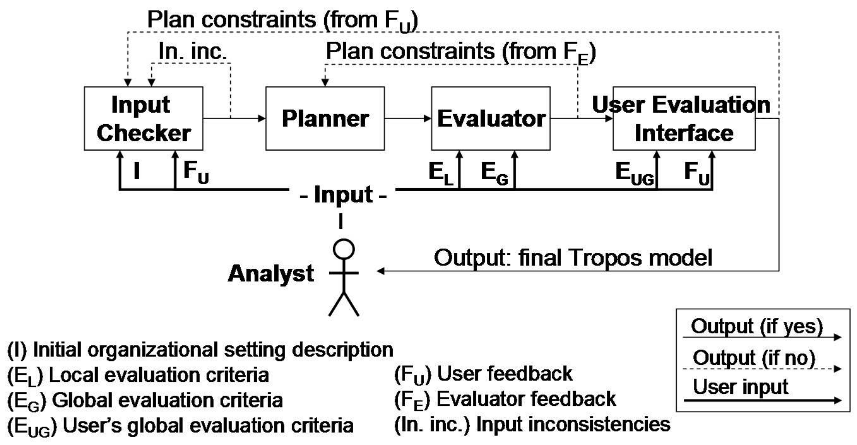
An example of a local evaluation criterium we consider in this paper, is related to workload distribution. We assume that each actor, either human or software, wants to minimize the number and complexity of goals it/he/she needs to satisfy. Complexity of a goal for an actor measures the effort required to achieve the goal. Complexity has to be defined explicitly for leaf goals, i.e. for those goals that could be assigned to actors that have capabilities to satisfy them. There is no need to define or calculate complexity for a goal that is to be further decomposed and delegated. Complexity is "local" in a sense that the same goal can have different complexity values for different actors. For each actor there is a maximum complexity it can handle, i.e. the sum of complexity values for all the goals this actor is assigned should be less than some predefined threshold, namely, maximum complexity. If this condition is violated the actor might be willing to deviate from the imposed assignment.

After the complexity and maximum complexity values are defined, the evaluation procedure, which preliminary version was presented in [3], is as follows.

1. A plan P is generated by the planner.
2. Plan complexity values for each actor are calculated, by summing up the complexity values for all the goals this actor is assigned.
3. Actors whose plan complexity values are greater than the corresponding maximum complexity values are identified.
4. One of these actors is selected, namely, actor a_{max} which has the maximum difference δ between plan complexity and maximum complexity values.
5. A subset of actions $P_{dev} \subset P_{a_{max}}$ is formed with the total cost greater or equal to δ, where $P_{a_{max}}$ denotes those actions of P in which a_{max} is involved.
6. The definition of the planning problem is changed in order to avoid the presence of actions contained in P_{dev} during the next planning iteration.
7. The procedure re-starts with the generation of a next plan.

4 Requirements Analysis: a General Schema

Our proposal is to structure the requirements analysis process to support a designer in constructing and evaluating requirements models. The general schema



Plan constraints (from F_U)
In. inc.
Plan constraints (from F_E)
Input Checker
Planner
Evaluator
User Evaluation Interface
I
F_U
E_L
E_G
E_UG
F_U
- Input -
Analyst
Output: final Tropos model
(I) Initial organizational setting description
(E_L) Local evaluation criteria
(E_G) Global evaluation criteria
(E_UG) User's global evaluation criteria
(F_U) User feedback
(F_E) Evaluator feedback
(In. inc.) Input inconsistencies
Output (if yes)
Output (if no)
User input

Fig. 3. Requirements analysis process: a general schema.

of the process is presented in Figure 3. A preliminary description of an organizational setting, which is provided by a designer in terms of actors, goals and social relations, is analyzed and iteratively improved so as to output a model that guarantees the fulfillment of stakeholder goals and is good-enough with respect to a number of criteria. In the following we give details on the process steps and their interrelations. Most of the process steps can be automated, however, the presence of a human designer is inevitable – the design process for socio-technical systems can be supported by tools but cannot be automated.

As a first step, it is checked whether there exist at least one assignment of goals to actors that leads to the satisfaction of top-level goals. **Input checker** analyzes the organizational setting, detects inconsistencies, and proposes possible improvements, which then are either approved, or rejected, or modified by the designer. In particular, it is checked whether available actors possess enough capabilities to collectively satisfy their goals, and whether the relationships between actors permit this to happen. To analyze actor capabilities means to check that for each goal it is possible to find an actor capable of achieving each of its AND-subgoals or at least one of its OR-subgoals. To analyze actor interdependencies means to check whether a goal can be delegated from an actor who wants to achieve it to an actor who is capable of achieving it. Given a network of delegations between the actors, it is checked whether there exists a path between two actors. In Section 5, we give details on analyzing and dealing with missing capabilities, while the second kind of analysis is not covered in the paper.

After the input is checked, the first possible alternative is generated by the **Planner** component, which exploits AI planning algorithms to search for a solution as described in Section 2.

An alternative generated by the planner is then assessed by the **Evaluator** with respect to a number of criteria. These criteria are defined by the designer and refer to the optimality of the solution either from a global perspective (e.g. assessing the overall security or efficiency), or from the local perspectives of stakeholders (e.g. assessing the workload distribution). Evaluation criteria and procedures were discussed in Section 3. If evaluation reveals that an alternative is not acceptable, then the **Evaluator** provides feedback to the **Planner** in order to formulate constraints for the generation of the next alternative. If no further alternative can be generated, the current description of an organizational setting is changed according to the constraints identified by the **Evaluator**, and then is analyzed by the **Input checker**, and so on, iteratively.

Note that the output of the evaluation process needs to be approved by a human designer. **User evaluation interface** presents the selected alternative to the designer together with the summarized evaluation results. Another task of this component is to provide the designer with the interface for giving his feedback on why the selected alternative does not satisfy him. On the basis of this feedback the constraints for the generation of the next alternative are formulated and forwarded to the **Planner**. The result of the application of the approach is a new requirements model, which is, ideally, optimal or, in practice, good-enough with respect to all the local and global criteria, and is approved by the designer. After obtaining one satisficable alternative it is possible to repeat the process to generate others, reusing already obtained constraints.

5 Input Analysis

The input to the proposed requirements analysis process is provided by the designer, and can be logically divided into the following two categories. Firstly, it is an organizational setting description, i.e. actors, their goals and capabilities, dependencies among actors, possible ways of goal refinements (decompositions of a goal into AND/OR-subgoals). The second category contains evaluation criteria, either qualitative or quantitative, e.g. preferences or cost bounds, as discussed in Section 3.

As it was stated in Section 4, the description of an organizational setting can be analyzed with respect to capabilities of available actor and actor connectivity. In this section the process of analyzing actor capabilities is explained, while actor connectivity analysis is outside the scope of this paper.

Firstly, a "capability tree" for each high-level organizational goal is constructed. A "capability tree" is based on a goal model [1], which organizes goals in a tree structure reflecting AND- and OR-decompositions. In addition, a list of actors capable of achieving a goal is associated with each leaf node of a tree. Nodes with no associated actors are marked as unsatisfiable with the goal which causes the problem included in the label (*unsat : goal*). Otherwise, a node is marked as satisfiable (*sat*). After that, labels are propagated bottom-up to the root goal according to the following rules. If all OR-subgoals or at least one AND-subgoal of a goal is marked *unsat*, then the goal is also marked *unsat*, and *sat* otherwise. When an *unsat* label is propagated, the goals that cause unsatisfiability problem are accumulated together; at the end the label might look as

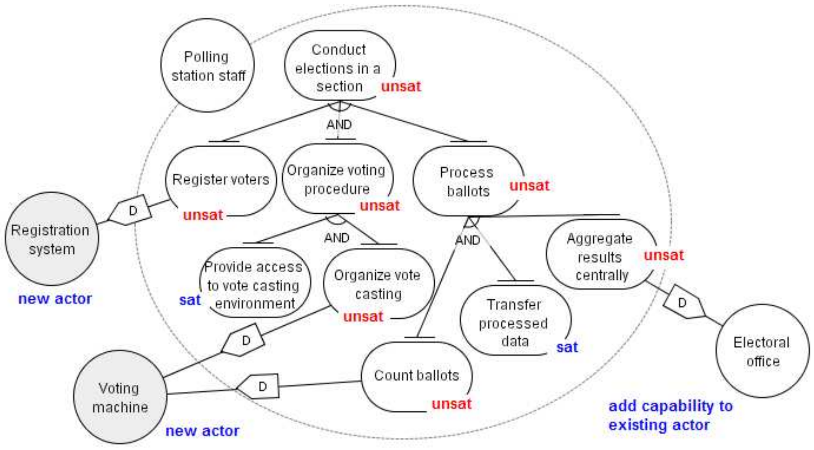


Fig. 4. Case study: conduct elections in a section.

unsat : $(G_1\ or\ G_2)\ and\ G_3$, meaning that to satisfy the root goal we need to find the ways to satisfy G_3 and either G_1 or G_2.

After a capability tree has been built and *sat*/*unsat* labels propagated, it becomes clear whether the root goal can be satisfied, and if not, what are the missing capabilities. There are two ways to deal with missing capabilities:

– Add a new capability to an existing actor. Such a decision could be based on the actual capabilities of this actor. Namely, if this actor is already capable of achieving one or several goals of the same type, it is likely that it could manage the new goal as well.
– If there is no way to add the missing capability to one of the existing actors, a new actor might be introduced.

6 E-Voting Case Study

The focus of the case study is on modelling requirements for an electronic based voting system that is to be introduced in the Autonomous Province of Trento by the next provincial elections (to be held in 2008). The Province is funding the ProVotE project [9], which has the goal of providing a smooth transition from the paper based voting system to new technologies. The project includes partners from the public administration, research centers and academia, and local industries.

6.1 Description

Voting in Italy consists of the following stages [8]: (a) Identification and registration of a voter at a polling station; (b) Casting a vote; (c) Counting votes and tabulating the results; (d) Transmission of the results to the offices responsible for data aggregation; (e) Sum and proclamation of the elected representatives.

Introducing the new technologies in all these stages not only changes the way in which votes are collected and processed, but also roles and responsibilities of the actors involved. One of the ProVotE activities was the extensive UML modelling of existing paper based voting procedures [8] in order to provide a baseline for the definition of new procedures. Obviously, there are many ways in which the paper based system might be transformed into an e-voting system, and thus, many alternative configurations should be analyzed and compared. However, UML modelling approach, being process-oriented, does not provide support for such an analysis. To complement UML modelling in tackling the above problems, Tropos modelling approach has been used, which provides a clear visual representation of the organizational setting (actors, goals, actor dependencies) combining the perspectives of different stakeholders[3].

Example 1. In Figure 4 the diagram representing the viewpoint of the *polling station staff* on the voting process is shown[4]. As presented in the diagram, the root level goal, *conduct election in a section*, is decomposed into the three AND-subgoals, two of which are then further decomposed. Some of the subgoals (*provide access to vote casting environment*, and *transfer processed data*) can be satisfied by the *polling station staff.* For the achievement of other goals the *polling station staff* depends on the other actors, either organizations, or humans, or software systems, e.g. the goal *register voters* is delegated to the *registration system* actor.

Example 2. The second example concerns the goal *transfer processed data*, and the *president of the section*, the leading member of the *polling station staff*, who is responsible for satisfying this goal, see Figure 5. Elections are conducted with the help of several *voting machines* installed in standard voter cabins in each polling station. A voter expresses his/her choice with the help of a touch-screen, and then a ballot is printed, approved by the voter and stored inside a machine. At the end of an election day, data from voting machines are collected on the special purpose USB keys, one per machine, and printed paper ballots are extracted from the machines (more details are given in [8]). Thus, there are three pieces of data that could be transferred from the *polling station* to the *Electoral office*, namely, USB keys, electronic data these keys contain, and paper ballots.

6.2 Illustrating the process

Capability analysis. Figure 4 depicts a capability tree for *conduct elections in a section* goal, in which two leaf goals are assigned *sat* labels, namely, *provide access to vote casting environment* and *transfer processed data*, meaning that

[3] See [2] for the discussion on why and how UML and Tropos modelling approaches are applied together in ProVotE.

[4] For the moment sat/unsat labels should be ignored, the necessary explanations will be given in Section 6.2.

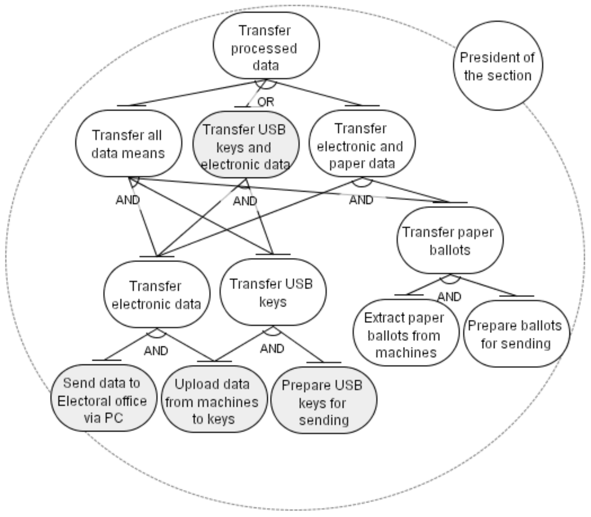

Transfer processed data
President of the section
OR
Transfer all data means
Transfer USB keys and electronic data
Transfer electronic and paper data
AND
AND
AND
Transfer paper ballots
Transfer electronic data
Transfer USB keys
AND
Extract paper ballots from machines
Prepare ballots for sending
AND
AND
Send data to Electoral office via PC
Upload data from machines to keys
Prepare USB keys for sending

Fig. 5. Case study: transfer processed data.

polling station staff possesses enough capabilities to satisfy both of them. After *sat/unsat* labels are propagated, it becomes clear that the goal *conduct elections in a section* cannot be satisfied due to the fact that the *polling station staff* is not able (or need extra support) to *register voters*, *count ballots*, and *aggregate the results centrally*[5]. To address the missing capability problems, the following steps are performed. Firstly, the capability to *aggregate the results centrally* is added to the existing actor, *Electoral office*; in the initial input setting this data could be omitted just by mistake, and such kind of input analysis helps to correct the errors. Secondly, two new actors are introduced to deal with *register voters* and *count ballots* goals, *registration system* and *voting machine*, respectively. These new actors are the parts of the new e-based voting system, specifying the requirements to which is what we aim at.

Planning and evaluation. In Table 1 the iterations of the planning and evaluation procedure applied to the case study are presented. When evaluating the workload, it is assumed that the goals *send data to Electoral office via PC* and *prepare USB keys for sending* are considered to be not very complex goals with the complexity value equal to 5 units for each of the two goals. The other three leaf goals are considered to be more complex with the complexity value equal to 10 units. For the sake of simplicity the complexity values for the same goal are the same for all actors. Maximum complexity values are the same for all the actors and are equal to 15 units.

[5] Extensions of *unsat* labels (which accumulate the goals that cause unsatisfiability problem as it was described in Section 5) are omitted for the sake of presentation simplicity.

#	**Description**	**Actor : Workload**	**Who deviates**
1	Depicted in Figures 5 and 6a.	President : **20**	President
2	President delegates the goal of *preparing USB keys for sending* to Secretary.	President : 15 Secretary : 5	Designer
3	Designer decides that to maintain the required security level all the data means should be transferred to the Electoral office. *Transfer all data means* goal is adopted, Secretary is in charge of *preparing USB keys for sending*, President takes care of all the other subgoals.	President : **35** Secretary : 5	President
4	The goal of *preparing paper ballots for sending* is delegated to Secretary.	President : **25** Secretary : 15	President
5	No plan can be generated, so actors' capabilities are analyzed, and it is inferred that Scrutineer can satisfy a goal of *extracting paper ballots from voting machines.* This goal is delegated to Scrutineer, see Figure 6b.	President : 15 Secretary : 15 Secretary : 10	—

Table 1. Case study: plans and their evaluation

```
OR_DECOMPOSES   PRES
      TR TRALL TREP TREK
AND_DECOMPOSES   PRES
      TREK TRE TRK
AND_DECOMPOSES   PRES
      TRE UPLOADONK SENDE
SATISFIES   PRES UPLOADONK
SATISFIES   PRES SENDE
AND_DECOMPOSES   PRES
      TRK UPLOADONK PREPAREK
SATISFIES   PRES PREPAREK
```

(a) Initial plan

```
OR_DECOMPOSES   PRES
      TR TRALL TREP TREK
AND_DECOMPOSES   PRES
      TRALL TRE TRK TRP
AND_DECOMPOSES   PRES
      TRP EXTRACTP PREPAREP
DELEGATES   PRES SCRU EXTRACTP
SATISFIES   SCRU EXTRACTP
DELEGATES   PRES SECR PREPAREP
SATISFIES   SECR PREPAREP
AND_DECOMPOSES   PRES
      TRE UPLOADONK SENDE
SATISFIES   PRES UPLOADONK
SATISFIES   PRES SENDE
AND_DECOMPOSES   PRES
      TRK UPLOADONK PREPAREK
DELEGATES   PRES SECR PREPAREK
SATISFIES   SECR PREPAREK
```

(b) Final plan

Fig. 6. Transfer data: the initial and the final plans.

7 Conclusions and Future Work

In this paper we have proposed a structured Tropos based requirements analysis process which focuses on exploring and evaluating alternative requirements models. Planning techniques are adopted for constructing alternative dependency networks, which are then evaluated with respect to a number of criteria. Evaluation criteria represent either a designer's point of view, or assess the models from the local perspectives of organizational actors. The approach is illustrated with the help of the case study taken from the ongoing project, which is about introducing e-based voting system in Autonomous Province of Trento.

The priority among the possible future work directions is on further implementation and testing the tool which supports all steps of the design process. Other important research directions include the further development of evaluation metrics, with a focus on local evaluation metrics; elaborating on the input analysis to address all the situations in which no solution is available; exploiting advances in AI planning by incorporating some of the evaluation metrics into the planning process; further experiments regarding the scalability and usability of the proposed approach.

8 Acknowledgements

This work has been partially funded by EU Commission, through the SENSORIA and SERENITY projects, by MEnSA-PRIN project, and also by the Provincial Authority of Trentino, through the MOSTRO project.

References

1. P. Bresciani, P. Giorgini, F. Giunchiglia, J. Mylopoulos, and A. Perini. Tropos: An agent-oriented software development methodology. *JAAMAS*, 8(3):203–236, 2004.
2. V. Bryl, F. Dalpiaz, R. Ferrario, A. Mattioli, and A. Villafiorita. Evaluating Procedural Alternatives.A Case Study in E-Voting. In *MeTTeG'07*, 2007.
3. V. Bryl, P. Giorgini, and J. Mylopoulos. Designing cooperative is: Exploring and evaluating alternatives. In *CoopIS'06*, pages 533–550, 2006.
4. V. Bryl, F. Massacci, J. Mylopoulos, and N. Zannone. Designing security requirements models through planning. In *CAiSE'06*, pages 33–47. Springer, 2006.
5. A. Dardenne, A. van Lamsweerde, and S. Fickas. Goal-directed requirements acquisition. *Science of Computer Programming*, 20:3–50, 1993.
6. S. Edelkamp and J. Hoffmann. Pddl2.2: The language for the classical part of the 4th international planning competition. Technical Report 195, University of Freiburg, 2004.
7. LPG Homepage. LPG-td Planner. http://zeus.ing.unibs.it/lpg/.
8. R. Tiella, A. Villafiorita, and S. Tomasi. Specification of the Control Logic of an eVoting System in UML: the ProVotE experience. In *CSDUML-06*, 2006.
9. A. Villafiorita and G. Fasanelli. Transitioning to eVoting: the ProVotE project and Trentino's experience. 2006. In EGOV-06.
10. D. S. Weld. Recent Advances in AI Planning. *AI Magazine*, 20(2):93–123, 1999.
11. E. S.-K. Yu. *Modelling strategic relationships for process reengineering.* PhD thesis, University of Toronto, 1996.