Requirements Engineering manuscript No.
(will be inserted by the editor)

Designing Socio-Technical Systems: From Stakeholder Goals to Social Networks

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

[1] University of Trento, Italy
[2] University of Toronto, Canada

The date of receipt and acceptance will be inserted by the editor

Abstract Software systems are becoming an integral part of everyday life influencing organizational and social activities. This aggravates the need for a socio-technical perspective for requirements engineering, which allows for modelling and analyzing the composition and interaction of hardware and software components with human and organizational actors. In this setting, alternative requirements models have to be evaluated and selected finding a right trade-off between the technical and social dimensions. To address this problem, we propose a tool-supported process of requirements analysis for socio-technical systems, which adopts planning techniques for exploring the space of requirements alternatives and a number of social criteria for their evaluation. We illustrate the proposed approach with the help of a case study, conducted within the context of an EU project.[1]

Key words socio-technical systems, exploring requirements alternatives, planning, evaluation metrics

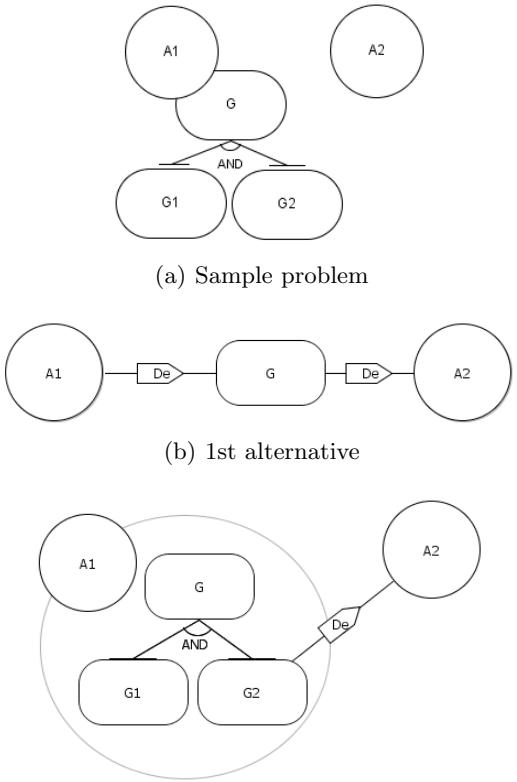
1 Introduction

Unlike their traditional computer-based cousins, socio-technical systems include in their architecture and operation organizational and human actors along with software ones, and are normally regulated and constrained by internal organizational rules, business processes, external laws and regulations [47]. Among the challenging problems related to the analysis and design of a socio-technical system is the problem of understanding the requirements of its software component, the ways technology can support human and organizational activities, and the way in which the structure of these activities is influenced by introducing technology. In particular, in a socio-technical system, human, organizational and software actors rely heavily on each other in order to fulfill their respective objectives. Not surprisingly, an important element in the design of a socio-technical system is the identification of a set of dependencies among actors which, if respected by all parties, will fulfill all stakeholder goals, the requirements of the socio-technical system.

Let us make the problem more concrete. KAOS [15] 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 Tropos [7], an agent-oriented software engineering methodology, which uses the i^* modelling framework [53] to guide and support the system development process starting from requirements analysis down to implementation. In Tropos and i^*, goals are explicitly associated with external stakeholders and can be delegated to other actors or the system-to-be. Or, they can be decomposed into subgoals that are delegated to other actors. Thus, requirements in Tropos and i^* are conceived as *goals associated to social actors within a network of social dependencies*. In this setting, selecting a set of assignments is more complex than in KAOS because delegations can be transitive and iterative. "Transitive" means that actor A_1 may delegate goal G to actor A_2 who in turn delegates it to actor A_3. "Iterative" means that an actor A_1 who has been delegated goal G, may choose to decompose it (in terms of

[1] The original publication will be available at www.springerlink.com in 2009.



(a) Sample problem

(b) 1st alternative

(c) 2nd alternative

Fig. 1 Sample problem: two alternative models

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 all of G (Figure 1b), or a part of it (Figure 1c). 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 "De"-labelled directed links. Even for such a simple example, the total number of alternative requirements models is large, 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 network 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 (Artificial Intelligence) planner which generates and evaluates alternative assignments of actor dependencies to identify an optimal design. We are exploring a range of measures for evaluating optimality, inspired by AI planning, multi-agent systems, as well as Economics. We are also experimenting with our prototype tool to evaluate its scalability to realistic design problems.

Specifically, our approach 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 allow them to fulfill all given goals.
- *Explore alternative dependency networks*: (a) Generate alternatives using a planning approach to construct an assignment 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.

We have presented the idea of adopting the AI planning approach for the problem of constructing design alternatives in [10,12]. In addition, in [10] we have discussed a preliminary approach to the evaluation of alternative system configurations. In [11], we place an emphasis on the *systematic process* of requirements analysis to support the design of socio-technical systems. In this paper, we integrate and extend earlier published work, and present a complete tool-supported process. The process combines formalization and analysis of an organizational setting, the use of planning techniques, and a set of criteria to evaluate the produced requirements models. Differently from our previous publications, we formalize the properties of the target requirements delegation network, and show how planning domain is defined so that these properties are satisfied by construction. Also, we present an extended set of evaluation criteria related to the cost and criticality of the obtained solution, as well as its relation to the degree of satisfaction of non-functional system requirements. We illustrate the approach with the help of an e-business case study. 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. In Section 2 we introduce the case study we use in this paper for illustrative purposes, elaborate on the problem definition, and present the general schema of the requirements engineering process which supports the designer in exploring and evaluating alternative options. In Sections 3 and 4 we explain in detail the two main components of the proposed approach, namely, the use of planning to construct alternative dependency networks, and a concrete set of evaluation criteria to evaluate and compare the constructed alternatives. Section 5 presents the re-

sults of application of the approach to the case study, and reports on scalability experiments. We conclude with a short overview of related work in Section 6, and summarizing remarks and future work directions in Section 7.

2 Requirements Analysis and Design in Tropos: Problems and Proposed Solution

In the following we present an e-business case study, and, with the help of it, discuss the problem of exploring and evaluating the space of design alternatives in the context of Tropos. Then, we present an overall schema of the requirements engineering process which addresses the above problem.

2.1 Case study

We further illustrate the problem on the basis of an e-business case study, adapted from the SERENITY EU project[2]. The case study focuses on the banking domain, namely, on the loan provision process. A detailed description of the case study is given in [45], while here for the sake of clarity and ease of understanding, we present only those details which are relevant to the paper.

The main actors of the e-banking scenario are the *customer*, the *BBB Bank* (hereafter *the bank*) and the *credit bureau*. The customer has an intention to buy a a house, and to do this, she needs to get a loan from a bank. When looking for a suitable proposal, the customer not only checks whether or not she is granted a loan (which might depend, e.g., on her employment status), but also on the conditions upon which the loan is given (e.g. the loan costs).

The diagram in Figure 2 presents the actors of the banking scenario along with their high-level goals and inter-dependencies. In the diagram, an arrow from *get a loan* to *find money* represents a means-end relation between two goals, meaning that whenever the source goal (means) is satisfied, the target goal (end) becomes satisfied.

Although the bank has a complex organizational structure and contains various departments, management hierarchy, and hundreds of employees, we consider only three roles (a role is a specification of an actor) within the bank played by its employees. In particular, we model and analyze strategic interests and functionalities of the bank in terms of those of the *bank manager*, the *junior clerk* and the *senior clerk*, which are represented as roles in Figure 2 and are connected to *BBB bank* actor with is-part-of relation. The manager's duty is leading the (local agency) of the bank. He can be involved in all steps of the loan approval process, but usually he is only involved when a final decision or supervision is needed. The junior clerk is a bank employee with just a couple of years of working experience, therefore he can deal only with those activities which require less skill and responsibility. The senior clerk is an experienced bank employee, who is responsible for performing all banking transactions for the customer.

The *credit bureau* is "a third-party business partner of a financial institution that processes, stores and safeguards credit information of physical and industrial agents" [45]. The bank, through the senior clerk or the bank manager, can contact the bureau in order to obtain the information about credit worthiness of the customer.

2.2 The problem: selecting a good enough alternative

According to Tropos methodology, the process of modelling and analyzing requirements for a socio-technical system is the following. Starting with the initial organizational setting presented in the diagram in Figure 2, the designer refines the goals and relations between the actors/goals in order to provide the details on how the loan approval is organized and how work is divided among existing and/or new actors. In particular, goal models are constructed for all top-level goals (*get a loan* in our case). A goal model [29] is a decomposition tree of a root goal into AND- or OR-subgoals. AND-decomposition of a goal refines the goals into subgoals which are to be satisfied in order to satisfy the parent goal, while OR-decomposition of a goal consists of a list of alternative subgoals any of which can satisfy the parent goal.

Given a goal model, the question is the following: *what is an optimal or, at least, good enough assignment of leaf goals to actors so that the root goal is satisfied when all actors deliver on the goals assigned to them?* The number of possible assignments is limited by the fact that different actors have different capabilities, and not all actors can delegate to all others. That is, an actor can delegate a goal only to actors he can depend on, e.g. those he knows, or manages, or pays money to. Still, the number of alternative delegation networks grows exponentially with the number of actors and/or goals that need to be dealt with. This calls for a way to automatically construct and evaluate possible alternatives to facilitate the design process.

To give an example, the corresponding problem to be addressed in the case study consists of finding the most effective way for bank employees to collaborate in order to satisfy customer requests and bank strategic interests. There could be several alternatives with respect to goal decomposition and assignment, depending on, for example, the involvement of the bank manager, or the division of labour between the junior and senior clerks. Another source of alternatives is the possible automation of (part of the) banking procedures. For this purpose, another system actor is introduced, the *bank internal computer system* (hereafter *the system*), which is capable of storing data, performing different calculations (e.g. of loan costs), etc. Different decisions on what

[2] http://www.serenity-project.org/

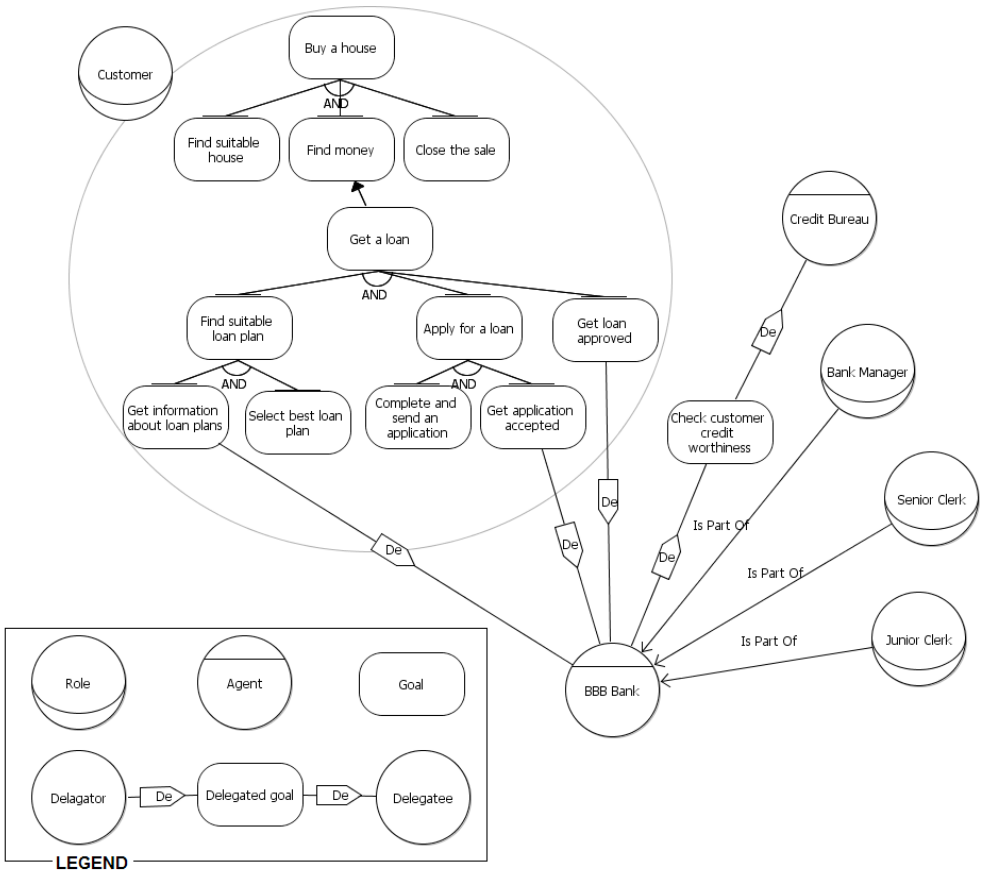


Fig. 2 E-business case study: initial settings

procedures to automate produce alternative social networks, which are all intended to satisfy customer needs but differ in terms of cost, risk, workload, etc.

The problem of exploring the space of alternative delegation networks is central to this paper. To address it, we complement the Tropos requirements analysis and design process as described in the following section.

2.3 The proposed requirements engineering approach

Our proposal is to structure the requirements analysis process to support a designer in constructing and evaluating requirements models. The general schema of the process, which we have already presented in [11], is presented in Figure 3. The description of an initial organizational setting is provided by a designer, and includes actors, their goals and capabilities, dependencies among actors, possible ways of goal refinements (decompositions of a goal into AND/OR-subgoals), and other goal and actor properties (see Section 3.1 for details). This input 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 user-defined criteria. Most of the process steps can be automated. However, the presence of a human designer (referred to as an analyst in the schema) is inevitable: the design of socio-technical systems can be supported by tools but cannot be automated.

As a first step, the process checks whether there exists at least one assignment of goals to actors that leads to the satisfaction of top-level goals. **Input checker** analyzes the organizational setting description, detects inconsistencies, and proposes possible improvements, which then are 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 who is 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, namely, whether there exists a path between two actors. In [11], we give details on analyzing and dealing with missing capabilities, which is based on label propagation algorithm similar to the one presented in [29]. After a missing capability is detected, there are two ways to deal with it:

– 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

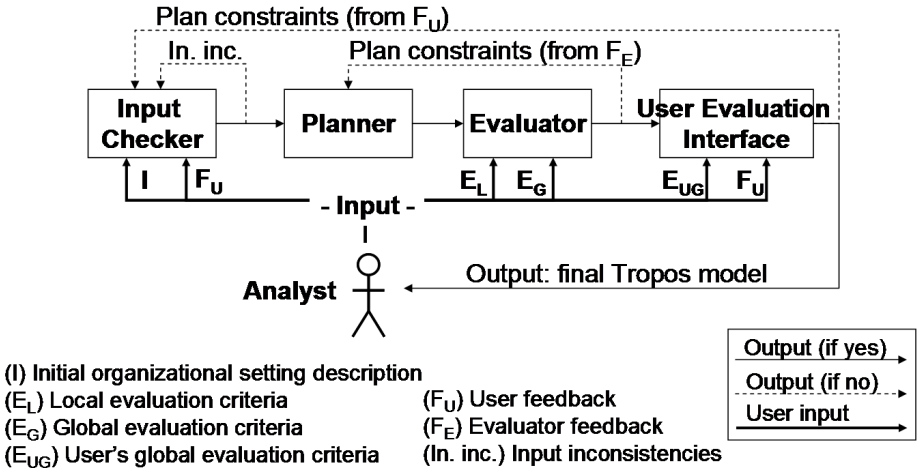


Fig. 3 Requirements analysis process: a general schema.

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.

After the input is checked, the first possible alternative is generated by the **Planner**, which exploits AI planning algorithms [52] to construct a social network that is capable of achieving the specified high-level system goals and, at the same time, satisfies a number of (optimality) criteria (see Sections 3-4 for the details).

An alternative generated by the planner is then assessed by the **Evaluator** with respect to a number of criteria. Some optimality criteria can be incorporated into the planning domain formalization (see Section 3.3), while the others are used a posteriori to evaluate an already obtained solution in terms of costs, risks, etc. These criteria are defined by the designer and refer to the optimality of the solution either from a global perspective (e.g. assessing the system security or efficiency), or from the local perspectives of stakeholders (e.g. assessing the workload distribution). Evaluation criteria and procedures are discussed in Section 4. 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. Also, it provides 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 this process 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. Note that after obtaining one satisficable alternative it is possible to repeat the process to generate others, reusing already identified constraints.

3 Generating Social Networks

In this section we give the details on how planning is used to support the designer in constructing networks of goal delegations among actors. We explain how the initial organizational setting and the desired plan properties are formalized, and how planning is conducted with the help of an off-the-shelf planning tool.

This section extends the framework initially presented in [10] in the following respects: the planning domain formalization is extended to include means-end, conflict and order relations among goals; the desired plan properties are formally defined and incorporated into the planning domain definition; the issue of domain preprocessing is addressed, which allows simplifying the domain for the better performance of the planning algorithm.

3.1 Formalizing the organizational setting

To model an initial organizational settings a designer needs to identify actors and goals, as well as social dependencies among actors. For this purpose, a set of first-order predicates is used, as presented in Table 1. The predicates take variables of three types: actor, goal and gtype (goal type) and are classified under the following categories:

Goal properties. To assign types to goals, type predicate is used. Goal types are used to group goals into domain specific types in order to allow for specifying the properties of an entire group of goals instead of specifying the same property for each goal separately. Goal refinements are represented using and/or_subgoal$_n$ predicates. The temporal order in which goals are achieved is constrained by the order predicate. To represent a goal conflict, i.e. the situation in which only one of several goals can be achieved, the conflict predicate is used. This predicate is symmetric, that is

$$\forall g_1, g_2 : \text{goal}\ \text{conflict}(g_1, g_2) \rightarrow \text{conflict}(g_2, g_1)$$

The means-end relation between goals is represented by the means_end predicate, which reflects the possibility that the satisfaction of an end goal is accomplished through the satisfaction of the corresponding means goal. When a goal is satisfied, the satisfied predicate becomes true.

Actor properties. Actor capabilities are described with can_satisfy and can_satisfy_gt predicates meaning that an actor has enough capabilities to satisfy either a specific goal, or any goal of a specific type. We define the following axiom for can_satisfy_gt predicate.

$$\forall a : \text{actor}, gt : \text{gtype}, g : \text{goal}\quad \text{can\_satisfy\_gt}(a, gt) \wedge \text{type}(g, gt) \rightarrow \text{can\_satisfy}(a, g)$$

Initial actor desires are represented with wants predicate.

Actor relations. Dependencies among actors are reflected by can_depend_on, can_depend_on_gt, and can_depend_on_g predicates, which mean that one actor can delegate to another actor the fulfilment of any goal, or any goal of a specific type, or a specific goal, respectively. We define the following axioms for can_depend_on and can_depend_on_gt predicates.

$$\forall a_1, a_2 : \text{actor}, gt : \text{gtype}\quad \text{can\_depend\_on}(a_1, a_2) \rightarrow \text{can\_depend\_on\_gt}(a_1, a_2, gt),$$
$$\forall a_1, a_2 : \text{actor}, gt : \text{gtype}, g : \text{goal}\quad \text{can\_depend\_on\_gt}(a_1, a_2, gt) \wedge \text{type}(g, gt) \rightarrow \text{can\_depend\_on\_g}(a_1, a_2, g).$$

Figure 4 presents an example of a formalized organizational setting corresponding to the scenario shown in Figure 2 (excluding information on roles and is-part-of relations).

Goal Properties
type(g : goal, gt : gtype)
and_subgoal$_n$(g : goal, g_1 : goal, . . . , g_n : goal)
or_subgoal$_n$(g : goal, g_1 : goal, . . . , g_n : goal)
order(g_1 : goal, g_2 : goal)
conflict(g_1 : goal, g_2 : goal)
means_end(g_1 : goal, g_2 : goal)
satisfied(g : goal)
Actor Properties
can_satisfy(a : actor, g : goal)
can_satisfy_gt(a : actor, gt : gtype)
wants(a : actor, g : goal)
Actor Relations
can_depend_on(a_1 : actor, a_2 : actor)
can_depend_on_gt(a_1 : actor, a_2 : actor, gt : gtype)
can_depend_on_g(a_1 : actor, a_2 : actor, g : goal)

Table 1 Formalizing an organizational setting: predicates

```
TBanking − gtype
Customer Bureau Bank Manager
    JuniorClerk SeniorClerk − actor
BuyHouse FindHouse FindMoney CloseSale GetLoan
    FindLoanPlan ApplyForLoan GetLoanApproved
    GetLoanPlanInfo SelectLoanPlan SendApplication
    GetApplAccepted CheckExtR − goal

(type GetLoanPlanInfo TBanking)
(type GetApplAccepted TBanking)
(type GetLoanApproved TBanking)
(can_depend_on_gt Customer Bank TBanking)
(can_depend_on Bank Manager)
(can_depend_on Bank Bureau)
(can_satisfy Customer FindHouse)
(can_satisfy Customer SelectLoanPlan)
(can_satisfy Customer SendApplication)
(can_satisfy Customer CloseSale)
(can_satisfy Bureau CheckExtR)
(and_subgoal3 BuyHouse FindHouse FindMoney CloseSale)
(and_subgoal3 GetLoan
    FindLoanPlan ApplyForLoan GetLoanApproved)
(and_subgoal2 FindLoanPlan
    GetLoanPlanInfo SelectLoanPlan)
(and_subgoal2
    ApplyForLoan SendApplication GetApplAccepted)
(means_end GetLoan FindMoney)
(wants Customer BuyHouse)
```

Fig. 4 An extract of formalization of the diagram in Figure 2

3.2 Planning approach

As indicated earlier, we adopt the view of *i** [53] and Tropos [7], where the requirements to socio-technical systems are conceived as networks of delegations among actors. Every delegation involves two actors, where one actor delegates to the other the fulfillment of a goal. The delegatee can either fulfill the delegated goal, or further delegate it, thus creating another delegation relation in the network. Intuitively, these can be seen as *actions* that the designer ascribes to the members of the organization and the system-to-be. Further, the task of constructing such networks can be framed as a *planning problem* where selecting a suitable socio-technical system structure corresponds to selecting a plan that satisfies the goals of human, organizational and software actors.

Thus, we have chosen an AI planning approach to support the designer in the process of selecting the best alternative socio-technical structure. Planning is about automatically determining a 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. to specify

- the initial state of the world,
- the desired state of the world,
- the actions that can be performed.

Once the domain is defined, the solution to the planning problem is a (not necessarily optimal) sequence of actions that makes it possible to reach a desired state starting from the initial state.

The set of predicates introduced in Table 1 is used to represent the initial state of an organizational setting. The desired state (or *goal of the planning problem*) is described through the conjunction of satisfied predicates: for each wants(a,g), satisfied(g) is added to the goal of the planning problem.

A plan, which is constructed to fulfill the goals of system actors, comprises the following actions:

- **Goal satisfaction.** Satisfaction of a goal is an essential action, which can be performed only by an actor who is capable of achieving the goal. The result of this action is the fulfillment of the goal. Action SATISFIES($a : actor, g : goal$) represents the fact that goal g is achieved by actor a.
- **Goal delegation.** An actor may choose to delegate one of his goals to another actor. We represent this transfer of responsibilities through action DELEGATES($a_1, a_2 : actor, g : goal$). It is performed only if the delegator wants the goal to be fulfilled and trusts that the delegatee will achieve it (i.e. he can actually depend on the delegatee). After having delegated the goal, the delegator is no longer responsible for its fulfillment, and does not care how the delegatee satisfies the goal (e.g. by his own capabilities or by further delegation).
- **Goal decomposition.** Two types of goal refinement are supported by the framework: AND- and OR-decomposition. A way/ways in which a goal can be decomposed is predefined and known to all the system actors. Actions AND_DECOMPOSES($a : actor$, $g, g_1, ..., g_n : goal$) and OR_DECOMPOSES($a : actor$, $g, g_1, ..., g_n : goal$) represent the fact that goal g is decomposed into n AND/OR-subgoals by actor a.

Actions are described in terms of preconditions and effects, both being conjunctions of formulas containing the predicates we have introduced, their negations, disjunctions, and universal and existential quantifiers. If a precondition of an action is true in the current state of the world, then the action can be performed; as a consequence of an action, a new state is reached where the effect of the action is true. Figure 5 presents a sample sequence of plan actions corresponding to the diagram in Figure 2.

```
AND_DECOMPOSES Customer BuyHouse
    FindHouse FindMoney CloseSale
AND_DECOMPOSES Customer GetLoan
    FindLoanPlan ApplyForLoan GetLoanApproved
AND_DECOMPOSES Customer FindLoanPlan
    GetLoanPlanInfo SelectLoanPlan
DELEGATES Customer Bank GetLoanPlanInfo
SATISFIES Customer SelectLoanPlan
AND_DECOMPOSES Customer ApplyForLoan
    SendApplication GetApplAccepted
SATISFIES Customer SendApplication
DELEGATES Customer Bank GetApplAccepted
DELEGATES Customer Bank GetLoanApproved
```

Fig. 5 A fragment of a plan corresponding to Figure 2

3.3 Desired properties of a solution

In this section, we discuss the properties a solution to a planning problem should satisfy. Later, we will define formally the four plan actions introduced above, so that these properties are satisfied by construction.

Let $I = (D, F)$ be the initial state of an organizational setting, where D is the domain, i.e. a set of objects (actors, goals and goal types), and F is a set of fluents, i.e. positive or negative grounded literals (predicates introduced in Table 1). We refer to the domain and fluent components of a state I as I_D and I_F, respectively. We also assume that

$$predicate(arg_1, .., arg_n) \in I_F \rightarrow \forall i \; arg_i \in I_D,$$

that is, if some predicate is in I_F then all the objects it takes as arguments are in I_D.

Let P be a plan, i.e. a (partially) ordered sequence of actions, G be a set of predicates which characterize the goal of a planning problem. Then, the execution of plan P in state I will result in a new state R of a socio-technical system,

$$R = result(P, I).$$

In this setting, a planning problem consists in constructing such a P that

$$G \subset R_F.$$

This property, referred to as **basic plan property** from now on, in our setting means that all goals of all actors are satisfied after a plan is executed. The property is formalized as shown in line 1 in Table 2. In this and further properties, the names of *actions* of a plan P are capitalized, while the names of *predicates* contain lowercase letters only.

A set of important plan properties, referred to as **plan compliance with the initial organizational setting**, consists in ensuring that a plan can actually be executed starting from the initial state of a socio-technical system. Namely, the following properties should hold.

1. Only an actor who has a capability to satisfy a goal is assigned the obligation to satisfy it.
2. A goal can be delegated from one actor to the other, only if the former can actually depend on the latter.
3. A goal can be AND/OR-decomposed only in a predefined way, that is, according to an AND/OR decomposition tree that is given as part of the problem statement.

For the formal representation of these properties see lines $2.1 - 2.3$ in Table 2.

Another related set of plan properties, referred to as **plan compliance with goal relations**, represent the fact that executing a plan cannot result in violating temporal, conflict, or means-end relations between goals.

To formalize these properties, let us define a *partial order* of plan actions[3]. Let a time step number $T(p) \in \{0, N-1\}$ be associated with each action $p \in P$ where $N \leq |P|$. Namely, p is performed at $T(p)$, which means that p's preconditions are true at $T(p)$, while the effects become true at $T(p)+1$. Also let $S(t), t \in \{0, N-1\}$ be the state of an organizational setting after all the preceding actions p, $T(p) < t$ are performed.

For a plan to be compliant with goal relations, the following properties should hold.

1. Goals should be satisfied in the correct order.
2. Only one of the conflicting goals can be satisfied in a plan.
3. If a means goal is satisfied and there are no satisfied goals which are in conflict with the end goal, then the end goal should become satisfied.

For the formal representation of these properties see lines $3.1 - 3.3$ in Table 2.

The last group of properties is concerned with the optimality of a plan, meaning that a plan should be such that the goals of system actors are satisfied in a minimal number of steps. In this respect, the following two properties, referred to as **non-redundancy properties**, should hold.

1. A goal cannot be satisfied more than once.
2. Plans are assumed to be minimal in the sense that any subset of the actions of a plan does not satisfy the goal for which the plan was intended.

For the formal representation of these properties see lines $4.1 - 4.2$ in Table 2. Note that the axiom 4.1 in Table 2 states that the same goal cannot be satisfied by different actors, while the fact that the same goal cannot be satisfied twice by the same actor is implicit in that P is a set, and thus, cannot contain duplicates. Also note that the second property implies the absence of delegation loops and unnecessary actions, meaning that no action over the same goal is performed twice by two different actors. Even though the first property can be inferred from the second one, we state both explicitly for the sake of clarity.

Now our aim is to define the plan actions in terms of their preconditions and effects in the way that the resulting plan satisfies all of the above four groups of properties.

3.4 Planning formalization

As we mentioned already, planning actions are described in terms of preconditions and effects, which in turn, are expressed in terms of predicates presented in Table 1. After the application of each subsequent plan action, the state of the world changes, namely, some of the fluents which describe the state of the target socio-technical system become true, while the others become false.

A plan which satisfies all actor goals, should step by step change the initial state into the final state where satisfied(g) is true if initially wants(a,g) was true for some actor a. Thus, the key idea is to propagate wants(..,g) along the delegation links from the actor who wants g initially, towards the actor(s) who are responsible for satisfying either g or its subgoals. With this idea in mind, we define the actions as presented in Table 3 and explained in the following.

- SATISFIES($a : actor, g : goal$). The preconditions of the satisfaction action are the following: a wants g to be satisfied, a is capable of achieving g (or any goal of the same type), all the goals that should be satisfied before g are satisfied, no goals which are in conflict with g are satisfied. The two effects are: g is satisfied, a no longer wants g to be satisfied. Thus, each satisfaction action removes (makes false) one wants predicate from the current state of the world.
- AND_DECOMPOSES($a : actor, g, g_1, ..., g_n : goal$) and OR_DECOMPOSES($a : actor, g, g_1, ..., g_n : goal$). The preconditions of a decomposition action are: a wants g to be satisfied, it is possible to AND/OR-decompose g into subgoals $g_1, ..., g_n$. As an effect, a no longer wants g to be satisfied, while $g_1, ..., g_n$ are added to the list of a's desires, i.e. a now wants $g_1, ..., g_n$ to be satisfied. Thus, a decomposition action removes wants predicate for the parent goal from the state of the world, but adds (makes true) n wants predicates, one for each of the subgoals.
- DELEGATES($a_1, a_2 : actor, g : goal$). The preconditions of a delegation action are: a_1 wants g to be satisfied, a can depend on a_2 either for any goal or for the goal of the type which g belongs to. The effects are the following two: a_1 no longer wants g to be satisfied, while g is added to the list of a_2's desires, i.e. a_2 now wants g to be satisfied. Thus, a delegation action removes one wants predicate from the current state of the world, and at the same time adds one: in a sense, it "passes" wants for g from the delegator to the delegatee.

[3] Most of the planning algorithms return a (partially) ordered sequence of actions as a resulting plan.

Basic plan property
1. $\forall a : actor, g : goal.wants(a, g) \in I_F \rightarrow$ $satisfied(g) \in R_F$
Compliance with initial organizational setting
2.1 $\forall a : actor, g : goal.SATISFIES(a, g) \in P \rightarrow$ $can\_satisfy(a, g) \in I_F$
2.2 $\forall a_1, a_2 : actor, g : goal.DELEGATES(a_1, a_2, g) \in P \rightarrow$ $can\_depend\_on\_g(a_1, a_2, g) \in I_F$
2.3.1 $\forall a : actor, g, g_1, \ldots, g_n : goal.AND\_DECOMPOSES(a, g, g_1, \ldots, g_n) \in P \rightarrow$ $and\_subgoal_n(g, g_1, \ldots, g_n) \in I_F$
2.3.2 $\forall a : actor, g, g_1, \ldots, g_n : goal.OR\_DECOMPOSES(a, g, g_1, \ldots, g_n) \in P \rightarrow$ $or\_subgoal_n(g, g_1, \ldots, g_n) \in I_F$
Compliance with goal relations
3.1 $\forall a : actor, g_1, g_2 : goal.order(g_1, g_2) \in I_F \rightarrow$ $satisfied(g_1) \in S_F(T(SATISFIES(a, g_2)))$
3.2 $\forall g_1, g_2 : goal.conflict(g_1, g_2) \in I_F \rightarrow$ $\neg(satisfied(g_1) \in R_F \wedge satisfied(g_2) \in R_F)$
3.3 $\forall p \in P,\ g_1, g_2 : goal.means\_end(g_1, g_2) \in I_F \wedge satisfied(g_1) \in T(p) \wedge$ $\neg(\exists g' : goal.satisfied(g') \in T(p) \wedge conflict(g_2, g') \in I_F) \rightarrow$ $satisfied(g_2) \in T(p)$
Non-redundancy properties
4.1 $\forall a : actor, g : goal.SATISFIES(a, g) \in P \rightarrow$ $\neg\exists a' \neq a : actor.SATISFIES(a', g) \in P$
4.2 $\neg\exists p.((R' = result(\{P \setminus p\}, I)) \wedge (G \subset R'_F))$

Table 2 Desired plan properties

SATISFIES$(a : actor, g : goal)$
precondition: $wants(a, g) \wedge can\_satisfy(a, g) \wedge$ $\forall g_{prev} : goal\ \neg(order(g_{prev}, g) \wedge satisfied(g_{prev})) \wedge$ $\forall g' : goal\ \neg(conflict(g, g') \wedge satisfied(g'))$ **effect:** $satisfied(g) \wedge \neg wants(a, g)$
AND/OR_DECOMPOSES$(a : actor, g, g_1, ..., g_n : goal)$
precondition: $wants(a, g) \wedge and/or\_subgoal_n(g, g_1, ..., g_n)$ **effect:** $\neg wants(a, g) \wedge wants(a, g_1) \wedge ... \wedge wants(a, g_n)$
DELEGATES$(a_1, a_2 : actor, g : goal)$
precondition: $wants(a_1, g) \wedge can\_depend\_on\_g(a_1, a_2, g)$ **effect:** $\neg wants(a_1, g) \wedge wants(a_2, g)$

Table 3 Actions: preconditions and effects

In addition to the above, the following rules should apply:

- When all AND-subgoals or at least one of the OR-subgoals of a goal are satisfied, then satisfied should become true for this goal.
- When a means goal of a means_end relation is satisfied and there are no satisfied goals which are in conflict with the end goal, then the end goal should become satisfied.

To address these one can define *axioms*, or *derived predicates* that hold in every state of the system and are used to complete the description of the current state [18]. However, due to performance problems introduced by derived predicates in practice, we postpone the discussion of this issue until Section 3.5.

Now, after the planning domain is fully defined, we need to show that **the desired plan properties** (see Table 2) **are guaranteed** by our approach.

Basic plan property. Following AI planning algorithms, a plan is constructed in such a way that the formula which represents the goal of the planning problem becomes true after the plan is applied to the initial setting. Thus, all satisfied(g) predicates which comprise

the goal of the planning problem are true after all the plan actions are executed.

Plan compliance with the initial organizational setting. According to the definitions of the plan actions (see Table 3):

- *satisfaction* action cannot be performed by an actor who has no capability to satisfy a goal;
- *delegation* action cannot be performed if the delegator cannot depend on the delegatee;
- *decomposition* of a goal can only be done in accordance with the AND/OR goal tree given for the goal.

Plan compliance with the goal relations. According to the definition of the plan actions (see Table 3) and domain axioms:

- a goal cannot be satisfied earlier than any goal it is in *order* with;
- a goal cannot be satisfied if another goal with which the former is in *conflict* is satisfied;
- satisfaction of a *means* goal implies the satisfaction of the corresponding *end* goal.

Regarding the satisfaction of **non-redundancy properties** there are two following observations. Firstly, for the absence of redundancy we rely not on the way our domain is defined, but on the planning approach itself. Basically, as most planning algorithms search for a (locally) optimal plan [52], a plan containing redundant actions can be immediately replaced by a better one (which is shorter and still satisfies the planning problem goal) just by removing a number of actions. Secondly, the property stating that each goal should be satisfied only once, is supported by the fact that once a goal is satisfied, satisfied predicate becomes true for it and it cannot be made false by any other action.

3.5 Implementing the planning domain

An important problem we have faced during the implementation of our approach, is the problem of choosing the "right planner" among off-the-shelf tools available. In the last years many planners have been proposed [40], which are based on different (classes of) algorithms, use different domain representation languages, adopt different heuristics for making the plan search efficient, etc. We have compared a number of planners (see [12] for the details) with respect to following requirements:

- As discussed earlier in this section, we do not want the planner to produce redundant plans. A plan is non-redundant if after deleting an arbitrary action, the resulting plan is no more valid (i.e. it does not allow reaching the desired state of the world from the initial state). Some planners we have tested (e.g. $DLV^{\mathcal{K}}$ [41]), do not satisfy this requirement.
- The planner should use PDDL (Planning Domain Definition Language) since it has become a "standard" planning language and many research groups work on its implementation. Moreover, the language should support a number of advanced features that are essential for implementing our planning domain (e.g. negation in a planning problem goal, which is not allowed in IPP [32]). Ideally, the planner should support the last stable version of PDDL, so PDDL3 [26] or at least PDDL 2.2 [18] should be supported.
- It is desirable that the planner is available on both Linux and Windows platforms as a set of the Tropos-based reasoning tools we have developed [44] work on both. However, we understand that most of the available planners are research tools, and so they are often released for the one specific platform only (with Linux being a more frequent choice).

Based on the above requirements, we have chosen LPG-td [36], a fully automated system for solving planning problems, supporting PDDL 2.2 specification language for implementing our planning domain. We do not claim that this is the best or the final choice of a planning tool, as the field of AI planning keeps developing. The use of PDDL as a domain representation language facilitates a lot any future transition from LPG-td to another PDDL-based planner.

Now let us discuss two implementation specific questions, namely, the implementation of decomposition actions, and the implementation of domain axioms.

AND/OR-decomposition actions should take $n + 2$ parameters, where n is the number of subgoals of the decomposed goal. As the number of action parameters in PDDL should be fixed, we have to fix an upper bound for n and introduce an and/or_subgoal_n predicate and a decomposition action for each $i \leq n$. According to our experience, large values of n is hardly the case in practice. E.g., for all the (medium-size industrial) case studies considered in [45] and [54], n is less or equal than 6. In Figure 6(b) an example of PDDL code of the AND-decomposition action for the case of two subgoals is presented.

As discussed in the previous section, **axioms** in our planning domain should be defined for the following cases:

- to infer the satisfaction of a goal from the satisfaction of all its AND-subgoals, or one of its OR-subgoals;
- to infer the satisfaction of an end goal from the satisfaction of its means.

Also, as explained in Section 3.1, the following rules should hold for the planning domain predicates:

- if goal g_1 is in *conflict* with goal g_2, then g_2 is in conflict g_1, that is, conflict relations between goals is symmetric;
- predicates defining actors capabilities and possible dependencies either for all goals or for the goals of a specific goal type (can_depend_on, can_depend_on_gt, can_satisfy_gt), should be "instantiated" so that only

```
(: actionSatisfies
   : parameters(?a − actor ?g − goal)
   : precondition(and
      (can_satisfy ?a ?g)
      (forall(?g1 − goal)(and
         (or (not(conflict ?g1 ?g)) (not(satisfied ?g1)))
         (or (not(order ?g1?g)) (satisfied ?g1))
         (or (not(means_end ?g1 ?g)) (not(satisfied ?g1))))
      )
      (wants ?a ?g)
   )
   : effect(and
      (satisfied ?g)
      (not(wants ?a ?g))
      (pr_satisfies ?a ?g)
   )
)
```

(a) Satisfaction action

```
(: actionAND_Decomposes2
   : parameters(?a − actor ?g ?g1 ?g2 − goal)
   : precondition(and
      (and_subgoal2 ?g ?g1 ?g2)
      (wants ?a ?g)
   )
   : effect(and
      (wants ?a ?g1)
      (wants ?a ?g2)
      (not(wants ?a ?g))
      (pr_and_decomposes2 ?a ?g ?g1 ?g2)
   )
)
```

(b) Decomposition action

Fig. 6 Samples of PDDL code.

capabilities and possible dependencies for the concrete goals (can_depend_on_g and can_satisfy) can be used in action formalization.

However, defining axioms in terms of *derived predicates* [18] increases the complexity of the planning problem to the point where the planner is not efficient anymore. This problem is neither new, nor specific to the planner we have adopted, as there is always a trade-off between the expressiveness and manageability of a formal domain definition language [25]. In a number of works [25,17,23] different approaches to the *preprocessing* of complicated planning domains are proposed. The key idea is to define a mapping that transforms a complicated domain to the equivalent one, in which axioms, quantifies, conditional effects and the like, are represented using a restricted (and less expressive) subset of PDDL language. For the domain axioms, the approach consists in introducing new domain actions and modifying the preconditions and effects of the existing actions.

None of the above cited approaches is suitable for dealing with the axioms in our planning domain. The algorithm presented in [25] was proven to be incorrect in [23]. The alternative algorithms presented in [23] and [17] consider only those axioms that cannot affect predicates which are changed in effects of any domain action. However, this is not the case in our domain, as satisfied predicate has to be changed both by actions and axioms. Inspired by the above approaches, we propose the following transformation of our planning domain to its equivalent version that does not represent axioms explicitly.

Firstly, we define a number of new planning actions, presented in Table 4. Combines actions appear in a plan each time the satisfaction of a goal should be inferred from the satisfaction of its subgoals. Infers actions support *means-end* relationship between goals. Note that in the preconditions of this action we have to check whether the end goal is in conflict with any of the goals satisfied so far. If so, the satisfaction of the end goal cannot be inferred as, in our framework, the conflict relation is stronger than the means-end relation.

In order to enforce the use of means-end relation and to avoid redundancy, an additional constraint should be added to the precondition of the satisfaction action. As presented in Figure 6(a), a goal cannot be satisfied if its satisfaction can be inferred using means-end relation. Note that the meaning of predicate $pr\_satisfies$ used in Figure 6(a), as well as of the other "trace" predicates, will be discussed in Section 4.4.

The last step concerns preprocessing the planning problem specification. Namely,

- for each conflict(g_1, g_2) predicate in the problem definition, we add (avoiding the duplicates) conflict(g_2, g_1);
- each can_depend_on is replaced by can_depend_on_g predicates for all goals g;
- each can_depend_on_gt is replaced by can_depend_on_g predicates for all goals g of the corresponding goal type;
- each can_satisfy_gt is replaced by can_satisfy predicates for all goals g of the corresponding goal type.

As a result, with LPG-td, producing a plan for the planning domain with derived predicates takes more than 100 times longer than producing the same plan with the preprocessed domain. This, basically, means that planning with the derived predicates does not scale, and cannot be used in the real-life domains. During the evaluation of a plan, which is discussed in the next section, Combine and Infers actions are not taken into considerations, as only those actions which represent a temporal act, rather than just an immediate inference, matter for the evaluation process.

The proposed requirements analysis approach is supported by S&D (Security and Dependability) Tropos Tool [44]. 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. Further work on the tool (mainly, the

COMBINES_AND$(g, g_1, .., g_n : goal)$
precondition:
and_subgoal$_n$(g, g$_1$, .., g$_n$) $\wedge$ satisfied(g$_1$) $\wedge$ .. $\wedge$ satisfied(g$_n$)
effect:
satisfied(g)
COMBINES_OR$(g, g_1, .., g_n : goal)$
precondition:
or_subgoal$_n$(g, g$_1$, .., g$_n$) $\wedge$ (satisfied(g$_1$) $\vee$ .. $\vee$ satisfied(g$_n$))
effect:
satisfied(g)
INFERS$(g\_means, g\_end : goal)$
precondition:
means_end(g_means, g_end) $\wedge$ satisfied(g_means)$\wedge$
$\forall$g : goal $\neg$(conflict(g, g_end) $\wedge$ satisfied(g))
effect:
satisfied(g_end)

Table 4 Additional actions: preconditions and effects

implementation of evaluation criteria discussed in the next section, the work on which has just been started) is among our primary future research directions.

4 Evaluating Social Networks

The alternative social networks generated by the planner need to be evaluated and approved by the designer. However, such an evaluation can be complex enough even for designers with considerable domain expertise, and thus a supporting tool would be beneficial. Alternative networks can be evaluated both from *global* and *local* perspectives, i.e. from the designer's point of view and from the point of view of an individual actor who is part of the network. In this section we discuss a number of evaluation criteria of both perspectives, as well as their application to the requirements networks.

4.1 Global evaluation criteria

The optimality of a solution in the global sense could be assessed with respect to the following criteria.

- Length of proposed plan.
- Overall plan cost.
- Degree of satisfaction of non-functional requirements.

Length of proposed plan. The number of actions in the proposed plan is most often the criterium for the planner itself to prefer one solution over another. For those planners which produce partially ordered plans (e.g. IPP [32]), the corresponding measure is the number of time steps in which a plan can be executed. Thus, it can be assumed that a plan produced by a planning tool is already optimal (or, in many cases, locally optimal, e.g. in case of LPG-td) in terms of length minimization.

Overall plan cost. In AI planning, criteria which take costs into consideration, are related to the idea of plan metrics introduced in PDDL 2.1 [20][4]. Plan metrics specify the basis on which a plan is evaluated for a particular problem, and are usually numerical expressions to be minimized or maximized. E.g. a cost or duration can be associated to a domain action, and then a plan is constructed so that the sum of costs/durations of all the plan actions is minimal. However, the complexity of the problem of optimizing a solution with respect to the defined metrics is high, in fact, "the introduction of numeric expressions, even in the constrained way that we have adopted in PDDL 2.1, makes the planning problem undecidable" [20]. Thus, effective use of plan metrics is still a research issue, and, as a consequence, the feature is poorly supported by the available planning tools.

Moreover, the planning metrics are not sufficient in cases where a cost should be assigned to *instances* of a planning action, while the planning metrics in PDDL 2.1 work on a generic rather than an instance level. Namely, we want to assign different costs to SATISFIES(SeniorClerk,CalcIntRating) and SATISFIES(Manager,CalcIntRating) actions, while planning metrics allow us to assign a cost value only to the "generic" action SATISFIES(a: actor, g: goal). This is not satisfactory, as in our planning domain, costs associated with *satisfying* the same goal are likely to be different for different actors, as well as costs associated with *delegating* the same goal to different actors can differ (e.g. delegating a goal to the colleague sitting next to you is often less costly than delegating the very same goal to

[4] Another related approach is planning with preferences (see e.g. [5]). However, for reasons similar to those for the planning metrics case, most plan evaluation criteria discussed in this section cannot be incorporated into planning with preferences.

the employee of another department who you even do not know in person). Thus, instead of using the planning metrics, we propose to evaluate a plan a posteriori, and then, on the basis of the evaluation results, provide additional constraints for the search for the next, better plan.

Let us now introduce notions of an action and plan cost, and a general schema we use further in the paper to represent and compare costs.

A cost associated to an instance of a domain action can incorporate a variety of factors, such as the time spent for executing the action, the amount of (different types of) resources used, the associated effort (related to complexity of the action), etc. Let $\mathbf{c}(p)$, where p is the instance of an action of the form $action\_name(arg_1, .., arg_n)$, be a cost vector associated to p, where $c_i(p)$ measures the i^{th} cost dimension.

For example, consider the satisfaction of the goal *calculate internal rating*. Let $\mathbf{c}(\mathsf{SATISFIES}(\mathsf{a}, \mathsf{CalcIntRating}))$, where a is the actor who actually satisfies this goal, be of the form $(total\_time, doc\_support, effort)$. Here the components of the cost vector are (a) the total time spent to satisfy the goal (i.e. to calculate an internal customer rating), (b) the number of instructions and regulations consulted in the bank electronic document base, (c) the subjective complexity that actor a attributes to the action (e.g. characterized as "high", "medium" or "low"). Then, two concrete examples for *manager* and *senior clerk* actors will be

$$\mathbf{c}(\mathsf{SATISFIES}(\mathsf{Manager}, \mathsf{CalcIntRating})) = (20', 3, low),$$

$$\mathbf{c}(\mathsf{SATISFIES}(\mathsf{SeniorClerk}, \mathsf{CalcIntRating})) = \\ = (35', 1, medium).$$

However, as the entries in a cost vector are, in general, heterogenous and measured in different units, this representation of cost is not satisfactory. Let us then introduce a scaling function $\mathbf{r_c}(\mathbf{c}(p))$, which maps heterogenous cost values to natural numbers from 1 to n, with 1 being the lowest and n the highest cost values. In the above example, let $n = 3$, and

$$[\mathbf{r_c}]_1(total\_time) = \begin{cases} 1, \, total\_time \leq 20', \\ 2, \, 20 < total\_time \leq 40', \\ 3, \, total\_time > 40', \end{cases}$$

$$[\mathbf{r_c}]_2(doc\_support) = \begin{cases} 1, \, doc\_support \leq 2, \\ 2, \, 2 < doc\_support \leq 5, \\ 3, \, doc\_support > 5, \end{cases}$$

$$[\mathbf{r_c}]_3(effort) = \begin{cases} 1, \, effort = low, \\ 2, \, effort = medium, \\ 3, \, effort = high, \end{cases}$$

where $[\mathbf{r_c}]_i(arg_i)$ is a "partial" scaling function for the i^{th} cost factor. In this setting,

$$\mathbf{r_c}(\mathbf{c}(\mathsf{SATISFIES}(\mathsf{Manager}, \mathsf{CalcIntRating}))) = (1, 2, 1),$$

$$\mathbf{r_c}(\mathbf{c}(\mathsf{SATISFIES}(\mathsf{SeniorClerk}, \mathsf{CalcIntRating}))) = (2, 1, 2).$$

The cost function of a plan P, called also *social cost* in networks literature (see e.g. [37]), is then defined either in the vector form

$$\mathbf{c}\,(P) = \sum_{p \in P} \mathbf{r_c}(\mathbf{c}(p)),$$

or in the scalar form

$$\bar{c}\,(P) = \sum_{p \in P} \frac{1}{n} \sum_{i=1}^{n} [\mathbf{r_c}(\mathbf{c}(p))]_i,$$

where $[\mathbf{v}(x)]_i$ is the i^{th} element of vector $\mathbf{v}(x)$.

Note that we do not discuss here neither the way the cost is measured along its various dimensions, nor the way the scaling function is constructed. These tasks are domain specific and require the interference of human experts.

Degree of satisfaction of non-functional requirements. By non-functional requirements (NFR) we mean quality criteria, which allow one to evaluate a socio-technical system, as opposed to functional requirements, which define the behavior of a system [14]. Examples of such criteria are efficiency, reliability, safety, usability, and many others. An important observation is that there can exist many alternative ways to implement system functional requirements, which differ with respect to those non-functional requirements that the designer considers important for a system. The "best" alternative may not exist, as often there are trade-offs between NFR, e.g. performance vs. safety, resilience vs. efficiency, etc. Therefore, the final choice of an appropriate alternative is usually up to a human designer.

To measure the impact of a design alternative on a certain non-functional requirement, either qualitative or quantitative metrics are used. *Qualitative metrics* are relative scales which allow comparing the degree of impact of two design alternatives towards a concrete NFR (e.g. such a metric may allow saying that "alternative X is better than alternative Y in terms of usability"). In goal modelling, qualitative reasoning on non-functional requirements typically means specifying whether the achievement of a goal contributes positively ("+") or negatively ("-") to a NFR. In [29] a formal framework for reasoning with goal models is introduced, which distinguishes between full and partial evidence of a goal being either satisfied or denied, and defines the propagation rules of the goal satisfaction labels. A number of works [30,51,35] use similar approaches to reason about the choice between functional alternatives on the basis of their contribution to a set of NFR.

In turn, adopting *quantitative metrics* means assigning numerical weights to quantify positive and negative influences of alternative design options on the degree of NFR satisfaction. In some cases, the numerical weights have a domain-specific physical interpretation (e.g. response or recovery times). However, as pointed

out in [35], most often it is not clear neither what these numbers mean (as they are subjective), nor where they come from (who is responsible of providing them).

Since the problem of both qualitative and quantitative evaluation metrics in the context of non-functional requirements in software design was studied by many researches (for examples see the references above), and a number of formal frameworks and tools were developed, we do not aim at inventing yet another way of dealing with NFR. A future work direction we consider, is incorporating an existing NFR reasoning technique into our framework.

4.2 Criticality of an actor in a plan

The *criticality* of an actor measures how a social network will be affected in case the actor has been removed from or has left the network. The notion of criticality is tightly connected to that of *resilience* of networks to the removal of their vertices in the social network literature [38]. A significant, while quite obvious related result is that most of the real-life and model networks are highly vulnerable to the removal of their highest-degree vertices, that is, the vertices with the highest number of in- and outgoing links.

In our framework, not only the links between the nodes matter, but also the goals that are assigned to the removed node according to the plan. In the following we discuss both criticality dimensions, that is, the impact of the removal of both satisfaction and delegation actions from a plan[5]. Note that as we consider only non-redundant plans, the removal of actor a *compromises* a plan P in case there exists an action $p \in P$ such that a is one of p's arguments, $a \in arg(p)$. Namely, if all actions a participates to are removed from a plan, the remaining actions do not satisfy all goals of the corresponding planning problem.

Leaf goals satisfaction dimension. When an actor is removed from a social network, all the leaf subgoals he was assigned in the plan remain unsatisfied. Namely, a measure of a node's a criticality in plan P is the (weighted) fraction of leaf goals that cannot be satisfied by P after a is removed from the social network constructed in accordance with P. Let an integer number $w(g)$ be the weight of goal g. Basically, $w(g)$ is the measure of importance of g for the socio-technical system defined by a human designer. Then the criticality of actor a in plan P is defined as follows:

$$cr_g(a,P) = \frac{\sum_{\mathsf{SATISFIES(a,g)}\in P} w(g)}{\sum_{\mathsf{SATISFIES(x,g)}\in P} w(g)},$$

where x is an *actor* and g is a *goal*.

[5] By action removal we mean, for instance, temporal unavailability of an actor or the failure of a communication link.

If all goals are considered equally important for the system ($\forall g, g' : goal\ w(g) = w(g')$), then the criticality can be calculated as follows:

$$cr_g(a,P) = \frac{|g.\mathsf{SATISFIES(a,g)} \in P|}{|(g,x).\mathsf{SATISFIES(x,g)} \in P|},$$

where x is an *actor* and g is a *goal*.

Dependency dimension. Together with an actor, a number of in- and outgoing dependencies for goals are removed from the network. This means that a number of delegation chains become broken and the goals delegated along these chains cannot reach nodes at which they will be satisfied (either directly or after being AND/OR-decomposed). Thus, we can define another set of measures of a node a criticality in plan P, namely, a fraction of "lost" dependencies (ingoing, outgoing, or any type) after a is removed from the social network constructed in accordance with P:

$$cr_{in}(a,P) = \frac{\sum_{\mathsf{DELEGATES(a',a,g)}\in P} w(g)}{\sum_{\mathsf{DELEGATES(x,y,g)}\in P} w(g)},$$

$$cr_{out}(a,P) = \frac{\sum_{\mathsf{DELEGATES(a,a',g)}\in P} w(g)}{\sum_{\mathsf{DELEGATES(x,y,g)}\in P} w(g)},$$

$$cr_{dep}(a,P) = cr_{in}(a,P) + cr_{out}(a,P),$$

where a', x, y are *actor*s and g is a *goal*.

In a number of other frameworks [53,24,8], each dependency link is assigned a certain criticality value, either qualitative or quantitative, which is the measure of how the system will be affected if this dependency fails. In i^* [53] the notion of dependency strength is defined, that is, a dependency is considered to be open (which corresponds to the lowest criticality level), committed, or critical. In [24], in- and outgoing criticality of an actor in a requirements model is defined as the sum of, respectively, the criticality of the in- and outgoing dependencies this actor participates. In addition, the authors present the procedure of complexity and criticality analysis (where complexity is the measure of effort required from an actor to satisfy a specific goal). Namely, the procedure identifies all the actors for which the values of complexity and criticality are greater than the respective predefined maximum values. Then, in order to reduce complexity and criticality of the existing actors, new (software) actors are introduced and the dependencies which violate complexity and criticality constraints are redistributed among these new actors.

In [8] the notion of security criticality is introduced, being the measure of how system security will be affected if the security constraint is violated. An example of a security constraint in our scenario would be to *keep customer's data private* associated to the goal *register customer*. The maximum value of criticality is defined for each actor, and an algorithm is proposed to reduce the criticality (as well as the complexity) of an overloaded actor by redistributing the goals and tasks

of this actor to others.

Actor criticality with respect to a set of goals. The proposed criticality measures consider the weighted fraction of leaf goals that are not satisfied or dependencies that are lost after an actor is removed from a network. However, it is also important to quantify the impact of a node removal on the top-level goals of a socio-technical system, or, in general, on any predefined set of non-leaf goals. To address this issue, the following measure is introduced.

Let $G_{aff}(a, P)$ be a set of goals affected by the removal of actor a from the dependency network constructed in accordance with plan P. Namely, if a is removed, all goals in $G_{aff}(a, P)$ cannot be satisfied by the above socio-technical structure. Also, let $G_{dir\_aff}(a, P)$ be the set of goals directly affected by the removal of a:

$$G_{dir\_aff}(a, P) = \{g : goal.\ \mathsf{SATISFIES(a, g)} \in P \vee \\ \exists a' : actor.(\mathsf{DELEGATES(a', a, g)} \in P) \vee \\ \exists a'' : actor.(\mathsf{DELEGATES(a, a'', g)} \in P)\}$$

Let G_r be the set of "reference" goals, i.e. top-level or any predefined subset of system goals with respect to which criticality of an actor in a plan will be evaluated. For example, in the e-business case study, G_r can consist of three goals, *get information about loan plans*, *get customer application accepted* and *get loan approved*, or, if one wants to evaluate the criticality of the system with respect to the loan approval process, G_r will consists of its three subgoals, that is, *evaluate loan*, *decide on loan* and *finalize the terms*.

To construct $G_{aff}(a, P)$ goal set corresponding to the set $G_{dir\_aff}(a, P)$, the modification of the label propagation algorithm [29] can be used, which allows inferring the (un)satisfiability of top goals by propagating through a goal graph the labels representing evidence for the goals being either satisfied or denied. Satisfiability is propagated bottom-up, from the leaf to the top-level goals, along decomposition, means-end and conflict relations.

The modifications of the algorithm consist in that the propagation starts not from the set of the leaf goals, but from the goals in $G_{dir\_aff}(a, P)$ (which are assigned unsatisfiability labels), and those leaf goals which are not the (recursive) AND/OR-subgoals of any goal in $G_{dir\_aff}(a, P)$. The unsatisfiability labels of goals $g \in G_{dir\_aff}(a, P)$ remain unchained even if their satisfiability is inferred by the label propagation algorithm. The latter may happen if in P goal $g \in G_{dir\_aff}(a, P)$ was delegated along the delegation chain which involved the removed actor a, while the satisfaction of g was performed by the actor(s) different from a. The above modifications work due to non-redundancy of the constructed plans, that is, it is assumed that P does not contain alternative ways to satisfy any of the system goals.

After $G_{aff}(a, P)$ goal set is constructed, the criticality of a in P with respect to G_r is defined as follows.

$$cr(a, P, G_r) = \frac{\sum_{g \in G_r \wedge G_{aff}(a,P)} w(g)}{\sum_{g \in G_r} w(g)}.$$

In many cases, the more even the load (in terms of assigned goals and delegations) of system actors, the lower the criticality of each actor. The problem of balancing the load distribution is considered in Section 4.4, where an evaluation and replanning procedure is introduced. The key idea of this procedure is to formulate the constraints for the construction of the next plan on the basis of the evaluation of the current plan. However, if the workload is balanced, but the subgoals of each goal $g \in G_r$ are distributed among a large fraction of system actors, the criticality of each actor is quite high as the removal of one actor will cause the failure of most of the goals in G_r. One of the approaches to the problem of leveraging high criticality is runtime replanning of those fragment of the social network in which criticality constraints are violated. Though we do not discuss it in this paper, such an approach is a feasible extension of our framework and constitutes another future work direction.

4.3 Local evaluation: game-theoretic insights

A challenging characteristic of requirements analysis for a socio-technical system is the presence of human and organizational actors. These actors can be seen as players in a game theoretic sense as they are self-interested and rational. This may mean that they are willing to minimize the load imposed personally on them, e.g. they want to constrain the number and the complexity of actions they are involved in[6]. In a certain sense non-human system actors are players as well as it is undesirable to overload them. Each player has a set of strategies he could choose from, e.g. he could decide to satisfy a goal himself or to pass it further to another system actor. Strategies are based on player capabilities and his relations (e.g. subordination, friendship, or trust, all represented as possible dependencies in our framework) with other human, organizational and software actors in the system.

We assume that each player ascribes a cost to each possible action, as discussed in Section 4.1. Then, it is possible to calculate the cost of a given alternative (or the outcome of the game) for the player by summing up the weights of the plan actions this player is involved in. For each player, minimizing this cost means maximizing the utility of the game. One of the key game theoretical concepts is that of an *equilibrium* [39], which defines

[6] This is not always the case, as sometimes actors may want the workload to be the maximum they can handle, e.g. in looking for the reward like salary increase or a the recognition of the boss/colleagues.

the set of strategies, one for each player, which none of independent rational players wants to deviate from. By playing an equilibrium each player maximizes his utility locally, given some constraints. For example, playing the Nash equilibrium means that no player can benefit when deviating from his equilibrium strategy given that all other players play the equilibrium.

However, in non-cooperative setting, there could exist Nash equilibria whose social cost (the sum of individual costs for all players) is much worse than the social cost in the globally optimal situation, called social optimum. To measure the impact of lack of cooperation and coordination on the system effectiveness, the notion of *cost of anarchy* has been introduced in [33]. Cost of anarchy is the ratio between the worst possible Nash equilibrium and the social optimum. There exist numerous studies on the theoretical bounds on the price of anarchy in the specific cases (e.g. [33,1]), as well as the attempts to design games, i.e. strategies and reward schemas, so that to encourage behaviors close to the social optimum (see e.g. [3,34], and, more generally, mechanism design theory [16]).

A substantial difficulty in applying game-theoretic ideas to our problem is that all actors of a socio-technical system need to work cooperatively in order to satisfy all initial organizational goals. Differently from classical non-cooperative game theory, where all players choose their strategies independently and simultaneously before the game, in our problem actor choices are closely interrelated. A player cannot independently change his strategy because the new action sequence will very likely be unsatisfactory, i.e. it will not be a solution anymore. So, to satisfy stakeholder goals it is necessary to impose an additional load on some other actors in order to compensate the load the deviating player tries to avoid. The actors on which this additional load is imposed might not be satisfied with the new solution, and will try to deviate from the strategy they were imposed, and so on and so forth. Thus, if one actor wants to deviate from the generated solution, the re-planning is needed to search for another alternative option, which is then evaluated, possibly, to be re-planned again. In the following section, we discuss a "planning-and-evaluation" procedure which aims at finding a good enough (rather than optimal) delegation and assignment socio-technical structure among the available alternatives.

4.4 Supporting local evaluation

In an ideal setting, the objective of our framework is to produce plans which are optimal with respect to both global and local evaluation criteria. However, choosing the optimum among all available alternatives, which are in the general case exponentially many, might not be feasible in practice. Moreover, as noted by Herbert Simon [46], what makes humans effective (in comparison to machines) is their ability to identify a *satisficing* design as opposed to an optimal one. Thus, our approach to optimization, or, more precisely, to looking for a satisficing solution consists in the following: For all the global and local criteria, thresholds are specified, and a plan that stays within these thresholds is considered to be good enough to be adopted.

In this section we present the revised procedure we devised for optimizing a plan with respect to the local criteria [10]. The problem of evaluating and improving a plan with respect to such global criteria as the overall plan cost and actor criticality, can be faced in an analogous way. We do not report here the details in order not to overload the paper, and focus on optimizing a plan with respect to the local criteria.

An example of a utility function we consider in this paper, is related to workload distribution. We assume that each actor, human, organizational or software, wants to minimize the number and cost of goals he is assigned. The cost of a goal, as it was discussed in Section 4.1, can incorporate a variety of factors, such as the time and effort required to achieve it, resources used, etc. Costs have 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 a cost for a goal that is to be further decomposed and delegated. Costs are "local" in a sense that the same goal can have different costs for different actors. For each actor there is a maximum complexity (in terms of cost) it can handle, i.e. the sum of costs for all the goals this actor is assigned should be less than a predefined threshold, namely, maximum complexity. If this condition is violated, the actor might be willing to deviate from the imposed assignment.

More precisely, for all actors a_i, $i = \overline{1,n}$ and all goals g_k, $k = \overline{1,m}$, where n and m are the number of actors and goals, respectively, the complexity values are defined:

- $\mathbf{cs}_{ik}$ is the complexity for actor a_i of satisfying goal g_k;
- $\mathbf{cr}_{ik}$ is the complexity for actor a_i of decomposing goal g_k;
- $\mathbf{cd}_{ijk}$ is the complexity for actor a_i of delegating goal g_k to actor a_j.

Here we assume that $\mathbf{cs}_{ik}$, $\mathbf{cr}_{ik}$, $\mathbf{cd}_{ijk}$ are of the form $\mathbf{r_c}(\mathbf{c}(p))$, $p \in P$, that is, they are "normalized" with the help of scaling function $\mathbf{r_c}(..)$ (see Section 4.1 for the details).

The cost of a given alternative P for actor a_i is calculated by summing up the costs of actions of P in which a_i is involved, and is denoted by

$$\mathbf{c}(P, a_i) = \sum_{\mathsf{DELEGATES}(\mathsf{a_i},\mathsf{a_j},\mathsf{g_k}) \in P} \mathbf{cd}_{ijk} + \sum_{\mathsf{DECOMPOSES}(\mathsf{a_i},\mathsf{g_k},\mathsf{g_{k1}},\ldots,\mathsf{g_{kl}}) \in P} \mathbf{cr}_{ik} +$$

$$+ \sum_{\mathsf{SATISFIES}(\mathsf{a_i},\mathsf{g_k}) \in P} \mathbf{cs}_{ik},$$

where $\mathsf{DECOMPOSES}(\mathsf{a_i}, \mathsf{g_k}, \mathsf{g_{k1}}, ..., \mathsf{g_{kl}})$ stands for the AND/OR-decomposition of g_k into l subgoals $g_{k1}, ..., g_{kl}$.

After the costs are computed, for each actor the conditions are defined upon which an actor decides whether to deviate from an alternative P or not. The conditions could be either one of the following, or both.

- Actor a_i whose predefined maximum complexity $\mathbf{c}^{max}(a_i)$ is less than $\mathbf{c}(P, a_i)$ is willing to deviate from P. This condition is the one we consider in the paper.
- Actor a_i whose predefined upper bound $\mathbf{cdev}^{up}(a_i)$ on cost deviation is less than $\mathbf{c}(P, a_i) - avg_i(\mathbf{c}(P, a_i))$ is willing to deviate from P.

After the costs and maximum complexities are defined, the *evaluation procedure* is organized as follows.

1. Plan P is generated by the planner.
2. Plan cost for each actor is calculated, by summing up the costs of all the action the actor is involved in.
3. Actors willing to deviate from the plan are identified, i.e. actors whose plan cost is greater than the corresponding maximum complexity.
4. One of these actors is selected, namely, actor a_{max} which has the maximum difference δ between plan cost and maximum complexity:

$$\delta(a) = \sum_{i=1}^{n} [\mathbf{c}(P, a) - \mathbf{c}^{max}(a)]_i.$$

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 of P_{dev} during the next planning iteration.
7. The procedure restarts with the generation of a next plan.

The process stops when a good enough solution is found, i.e. no actors are willing to deviate from it and the designer approves this solution.

Step 6 of the procedure deserves additional explanations. In order to make it possible to avoid the actions contained in P_{dev} in the next plan, we introduce the following "tracing" predicates:

$\mathtt{pr\_satisfies}(\mathtt{a} : \mathtt{actor}, \mathtt{g} : \mathtt{goal})$,
$\mathtt{pr\_and/or\_decomposes}(\mathtt{a} : \mathtt{actor}, \mathtt{g_1}, \mathtt{g_2}, ... : \mathtt{goal})$,
$\mathtt{pr\_delegates}(\mathtt{a_1}, \mathtt{a_2} : \mathtt{actor}, \mathtt{g} : \mathtt{goal})$,

which become true when the corresponding action takes place. Then, if, for instance, P_{dev} contains a satisfaction action for goal g, the following line is added to the goal of the planning problem: $\mathtt{not\ pr\_satisfies}(a_{max}, g)$, which means that the next generated solution cannot contain this action.

5 Evaluation Results

In this section, we report on the application of the proposed approach to the e-business case study as well as to a number of other case studies, and then present the results of scalability experiments we have conducted to justify the use of a planning approach for medium size real-life case studies.

5.1 Case studies

In the early requirements model of the e-business case study in Figure 2, the goals *get information about loan plans*, *get application accepted* and *get loan approved* are delegated by the *customer* to the *bank*. To satisfy these goals, the actors representing the bank, i.e. the *manager* and the bank *clerks*, decompose these goal as follows. The first goal, *get information about loan plans*, can be satisfied either via online information request processing, or by allowing a customer to come and ask for the related information in person. The second goal, *get application accepted*, is decomposed into two OR-subgoals, one of which concerns the processing of hand-filled loan applications, while the other refers to the processing of online applications. In both cases, the customer and, after that, her request for a loan are registered. We assume that the bank would like to stick to either online or in-person way of working with customers, that is, conflict relations are defined (a) between goals *provide loan information in person* and *process online loan application*, and (b) between goals *provide loan information on online request* and *process hand-filled application*.

The third goal, *get loan approved*, is decomposed into three AND-subgoals, *evaluate loan*, *decide on loan* and *finalize the contract*. The first subgoal concerns the evaluation of credit worthiness of both new and existing bank customers, and, in the latter case, can be done either from scratch or using the previous evaluations for the same customer stored in the internal database. In turn, evaluating the customer credit worthiness from scratch, requires the involvement of the credit bureau. The goal of *finalizing the terms* concerns communicating the decision and available options to a customer (in person or via phone call), and, finally, signing the loan contract. Note that there are *order* relations among the goals, e.g. loan evaluation should be done before the final decision can be made and contract signed. We do not report all *order* relations here, though they are part of the planning problem file that is used in the experiments reported below.

All the actors, goals, possible ways of goal decomposition, conflict and order relations are formalized in the problem definition file, which format is the same as in Figure 4 and which we do not present here for the space reasons. Also, in this file possible dependencies among actors and actor capabilities are specified. Namely, man-

ager and senior clerk are capable of *calculating internal ratings*, *reviewing existing ratings*, *deciding on loans* and *signing contracts*. Senior and junior clerks are capable of *approving online information requests and applications*, *registering customers and applications* and *preparing contracts*. Manager, senior and junior clerks are capable of *providing information on loan plans in person* and *communicating final decision and contract options to customers by phone or in person*. The goal *check external rating* can be satisfied by the credit bureau. The software system, a new actor introduced to the scenario in the process of requirements analysis, is capable of achieving a number of technological goals, such as *processing online information requests*, *registering customers and online loan requests* and *checking bank database for the existing customer ratings*.

In total, the definition of the planning problem for the e-business case study comprises 52 entities (6 actors, 5 goal types and 41 goals organized in 7 decomposition levels), 91 predicates before and 132 predicates after the preprocessing (see Section 3.5 for the details on the preprocessing of planning problem specification).

Here we illustrate the planning and evaluation procedure presented in Section 4.4. We assume that only satisfaction actions have non-zero complexity for all the actors, and the complexity of one subgoal is equal to one unit for any actor. Maximum complexities are defined for the *manager* (1 unit) and for the *senior clerk* (3 units). In Table 5 the iterations of the planning and evaluation procedure applied to the case study are presented. Five iterations are required to reach a good enough solution. For each iteration we give a short textual description, the costs of the plan constructed on this iteration for each of the actors, and the name of a_{max} actor (see Section 4.4 for the details), who initiates the deviation from the generated plan as his complexity thresholds are violated. The final alternative is presented in Figure 7.

On each iteration, LPG-td took about 5.5 seconds to produce a plan. The plan corresponding to the final alternative consists of 57 actions organized in 33 time steps: LPG-td planner produces partially ordered plans, that is, actions that can be performed in parallel are grouped together.

Let us now discuss the evaluation of the final design alternative with respect to the global criteria presented in Section 4.1.

There can be identified at least two important *non-functional* aspects relevant for the e-business case study. The first aspect concerns *customer satisfaction*, which includes *quality of service* (referred to the direct communication between customers and bank clerks), *usability, or ease of use of the online system*, *availability* both of clerks and the online system, and *security of the banking system* as perceived by the customer. The second aspect refers to *maximizing bank profits*, which includes *reliability of customer credit trustworthiness assessment*, *security of the banking procedures*, and *attractiveness of bank services for the customer*. Both lists are not supposed to be exhaustive, other non-functional aspects can be added after examining the stakeholder interests and the specifics of a concrete bank.

To give an example of comparing two design alternatives with respect to a non-functional requirement, consider the two alternative ways the loan contract terms are communicated to the customer: either in person, or via phone. The customer may consider the former alternative to contribute positively to the security of the procedure just because he perceives personal communication to be more secure for such a private case as the loan contract discussion. Some of the listed non-functional aspect can be addressed only during the further elaboration of the case study. For instance, the questions related to usability and availability of the online banking system are considered during the detailed design of the latter. Some other aspects, such as the availability of the clerks, can be assessed only on the *instantiated* model, in which roles are assigned to concrete agents, and additional instance level constraints are enforced. An example of such constraint, related to security of banking procedures and reliability of customer rating calculations, is the requirement that certain phases of the customer assessment cannot be performed by the same clerks. To a certain extent, this constraint is already addressed at the level of roles, as in the final plan the *manager* takes the final decision on a loan, while the other operations are performed by the *senior clerk*. The problems related to requirements models instantiation and the analysis of the instance level constraints appear to be an interesting research direction and are among our ongoing activities.

The examples of the cost-based global evaluation criteria relevant to the case study are the ones concerned with the resources consumed in each of the alternative configurations of a socio-technical system. For instance, it might be of interest to assess how intensively the printer is used during the loan approval process (the result can be used to adjust the existing infrastructure), or how often the internal document base is consulted during the various stages of the loan approval process (this might be related to the unsatisfactory performance of the bank clerks, or may reveal the insufficient throughput of the internal network). It is relatively straightforward to quantify such criteria and to evaluate the alternatives using the cost-based schema presented in Section 4.1. Note that, as in case of non-functional requirements, some of the cost-based criteria are applicable only at the instance level, for example, in a model containing roles but not concrete agents it does not appear possible to assess the total effort (e.g. in working hours) normally spent for processing one customer application.

The scale of the case study, at least as it is presented in the paper, does not provide the opportunity to fully test the criticality evaluation metrics discussed in Section 4.1. The observations that can be made on the final design alternative presented in Figure 7 with respect

it#	Description	Actor : Workload	Who deviates
1	Manager answers customer requests on loan plans, provides internal ratings and decides on loans; Junior clerk registers customer applications; Senior clerk communicates final decisions to customers and finalizes contracts.	Manager : 3 Senior Clerk : 3 Junior Clerk : 2 System : 0	Manager
2	Manager decides on loans; Senior clerk provides internal ratings, communicates final decisions to customers and finalizes contracts; customer requests are processed and customer applications are registered automatically by the System and approved by Senior clerk.	Manager : 1 Senior Clerk : 6 Junior Clerk : 0 System : 3	Senior Clerk
3	No solution found; $P_{a_{max}}$ of the previous iteration is revisited and SATISFIES(SeniorClerk, CalcIntRating) is replaced with SATISFIES(SeniorClerk, PrepareContract).	—	—
4	Manager decides on loans; Senior clerk processes customer requests and registers customer applications, provides internal ratings and signs contracts; Junior clerk communicates final decisions to customers and prepares contract templates.	Manager : 1 Senior Clerk : 4 Junior Clerk : 2 System : 0	Senior Clerk
5	Manager decides on loans; Senior clerk provides internal ratings, communicates final decisions to customers and signs contracts; customer requests are processed and customer applications are registered automatically by the System and approved by Junior clerk; Junior clerk prepares contract templates.	Manager : 1 Senior Clerk : 3 Junior Clerk : 3 System : 3	—

Table 5 E-business case study: plans and their evaluation

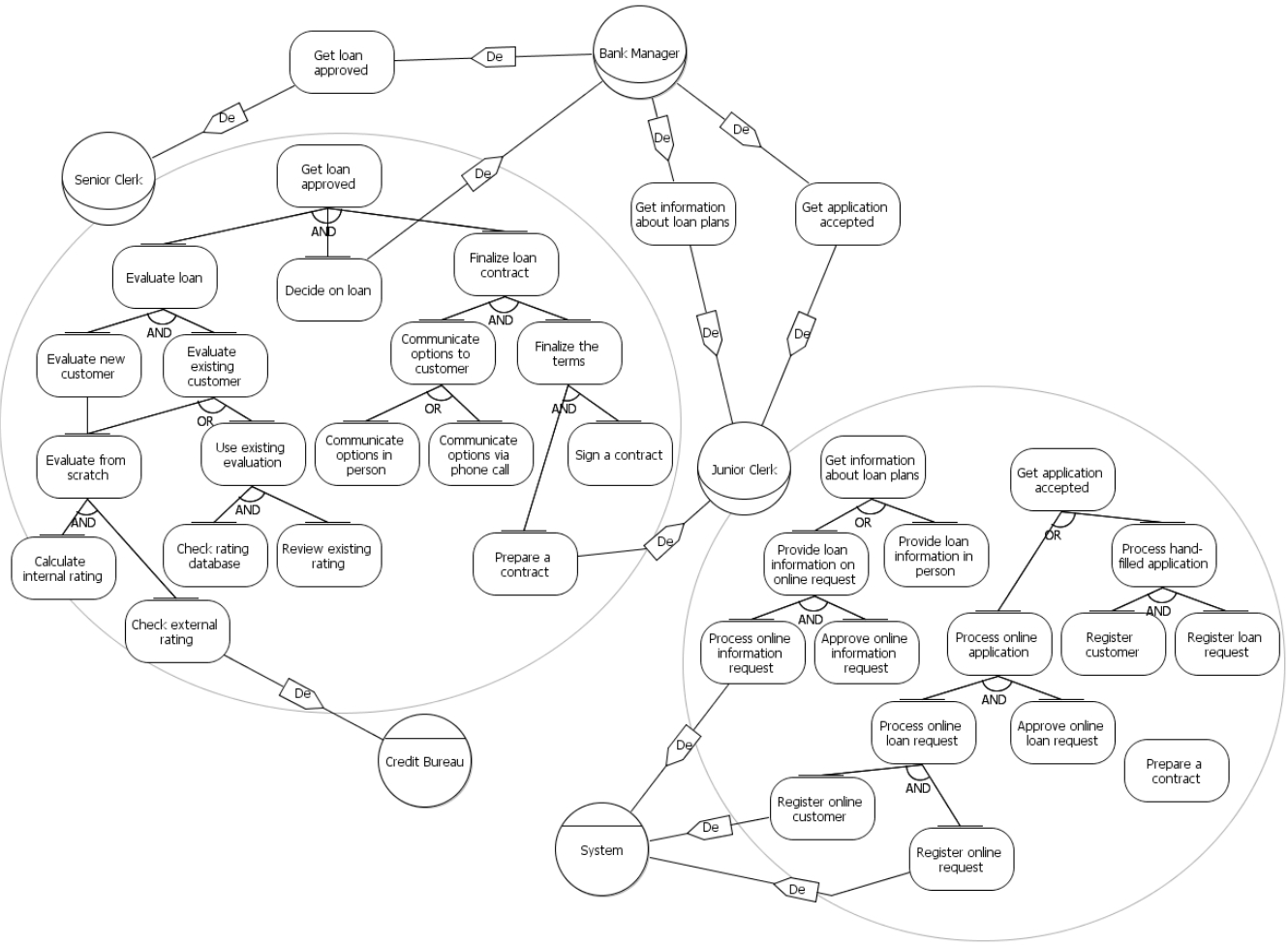


Fig. 7 E-business case study: adopted solution

to criticality are, for instance, the following. The *senior clerk* is a critical actor for the goal *get loan approved*, while the unavailability of the *manager* will not be highly critical, as the only contribution of the *manager* to this goal is making the final decision. In an alternative system configuration, in which the final decision is taken by the *senior clerk*, the criticality of the *manager* for the goal *get loan approved* is zero, however, as it was discussed above, this contributes negatively to the non-functional requirement for the security and reliability of the loan approval procedure.

To conclude this section, we mention a number of applications of the presented approach to various cases studies. In [12] we have customized our planning-based framework for the domain of secure system design. The planning domain is defined to guarantee that the resulting socio-technical model satisfies the trust and permission constraints imposed on it (e.g., no goal is delegated along an untrusted link). The framework is applied to the Medical Information System case study, which deals with the payment for medical care. Another extension of our framework [4] uses risk-based evaluation metrics for selecting a suitable design alternative, and aims at safety critical applications. The approach is evaluated on the basis of Air Traffic Management case study which comes from SERENITY Project [45]. The illustrative examples used in [11] are taken from the e-voting case study related to the project funded by the Autonomous Province of Trento, which has the goal of providing a smooth transition from the paper-based voting system to new technologies.

5.2 Scalability experiments

The case study considered in the previous section allows us to illustrate how the proposed planning-based framework works, and what support it provides to a designer. Still, a reader might wonder what happens in case of much bigger models, i.e. what conclusions can be drawn about the scalability of the approach. In this section, we report on a number of experiments related to the above question.

The main idea of the first part of the experiments was to understand how the growing complexity of a planning problem influences the performance of the approach. We have looked at series of planning problems with (a) growing number of goals to satisfy, and (b) growing complexity of the goal trees. All experiments were conducted using LPG-td planner [36].

A building block of each planning problem file is an *elementary tree* which contains 4 decomposition levels, 15 goals (G_i, $i = \overline{1,15}$), 2 OR and 4 AND decomposition relations:
(or_subgoal2 G1 G2 G3)
(and_subgoal3 G2 G4 G5 G6)
(or_subgoal2 G4 G9 G10)
(and_subgoal2 G5 G11 G12)
(and_subgoal2 G3 G7 G8)
(and_subgoal3 G7 G13 G14 G15)

All problem files contain 6 actors (A_i, $i = \overline{1,6}$) organized into three levels with respect to the relations between them:
(can_depend_on A1 A2)
(can_depend_on A1 A3)
(can_depend_on A2 A4)
(can_depend_on A2 A5)
(can_depend_on A3 A5)
(can_depend_on A3 A6)

Overlapping capabilities are introduced, namely, each leaf goal can be satisfied by two actors.

In the experiments reported in Table 6 the number of elementary goal trees N_{trees} which actor A_1 wants to satisfy was increasing. That is, we studied what happens when the planning problem grows "in breadth", namely, how the planner behaves in case the number of top goals to be satisfied increases, while the number of levels in goal decomposition trees stays the same. N_{elem} stands

N_{trees}	N_{elem}	N_{leafs}	t_{total}
1	15	4	0.14
2	30	8	1.08
3	45	12	5.24
4	60	16	7.43
5	75	20	6.03
6	90	24	15.3
7	105	28	13.25
8	120	32	16.83
9	135	36	19.83
10	150	40	26.52
11	165	44	37.29
12	180	48	ERR

Table 6 Experimental results: increasing the number of elementary goal trees

N_{lev}	N_{elem}	N_{leafs}	t_{total}	t_{total} with no OR's
4	15	4	0.15	0.14
7	43	7	4.39	4.31
8	51	11	8.88	8.71
9	59	15	6.07	5.48
10	67	19	7.93	7.65
11	75	23	5.94	5.95
12	83	27	12.91	11.84
13	91	31	10.49	8.07
14	99	35	12.58	9.3
15	107	39	12.59	10.48
17	115	43	15.11	10.13
19	123	47	21.61	17
23	131	51	40.39	34.66
24	139	55	ERR	ERR

Table 7 Experimental results: increasing the number of goal tree levels

for the total number of goals in the planning problem file, N_{leafs} stands for the number of leaf goals satisfied in a plan. t_{total} is the time (in seconds) the planner took to solve the problem, namely, it is the sum of two components, parsing and search time, the latter one being insignificant in all the experiments. *ERR* denotes the situations in which the planner was not able to produce a solution due to the problem complexity. As shown in the table, the planner was able to solve the problems with up to 11 top goals to satisfy (remember that each top goal is an elementary tree containing 15 goals organized into 4 decomposition levels).

In the experiments reported in Table 7, the number of goal tree levels N_{lev} for a top goal A_1 wants to satisfy was increasing. That is, we study what happens when the planning problem grows "in depth", namely, how the planner behaves in case the number of top goals (one goal in our case) to be satisfied does not change, while the number of levels in goal decomposition tree, and thus the number of subgoals increases. The meaning of N_{elem}, N_{leafs}, *ERR*, and t_{total} is the same as in Table 6. The first line of numbers in the table refers to the problem file containing one elementary tree, the second line – to the problem file containing three elementary trees, one at levels 1-3, and the other two at levels 4-7. For each of the subsequent lines a problem file was constructed by adding one or two levels of leaf goals to the previous problem file. As shown in the table, the planner was able to solve problems with the top goal having up to 23 decomposition levels.

The last column of values in Table 6 refers to t_{total} *with no OR's*, which is the time the planner took to solve the modified problem, where all OR-decompositions in a problem file were changed to AND-decompositions. The fact that t_{total} *with no OR's* is less than t_{total} in all the lines is not surprising, as each OR-decomposition doubles the number of alternative solutions, and thus, increases the search space. Additional line of experiments, which is not reported here, revealed that avoiding overlapping capabilities (i.e. reducing a problem file to the one in which each goal can be satisfied by just one actor) did not have a significant impact on t_{total} values.

The results reported in Tables 6 and 7, in our opinion, justify the scalability of the use of planning in the domain of requirements engineering. According to our experience, requirements models of real-life case studies (including the one considered in this paper) stay within the complexity limits which our planning approach can handle. For instance, for all the case studies reported in [45], the depth of goal decomposition trees is not greater than 10 (while in our approach the planner was able to process the goal tree with depth equal to 23). The number of top goals for the models in [45] is lower than 11, a number of elementary goal trees the planner was able to process in the experiments reported in Table 6.

The second set of experiments, which included not only planning but also the evaluation step, was conducted in order to study the scalability of the whole approach. Based on the obtained results, the following two observations were made. Firstly, parsing and search times were not influenced by additional predicates (negations of tracing predicates, see Section 4.4 for the details) in the goal of the planning problem. Secondly, convergence to a good enough plan appeared to depend on the ratio of acceptable (in terms of costs) assignments of goals to actors among all possible assignments. For relatively small problems (with 6 actors, 10 leaf goals in a plan, each goal can be satisfied by two actors) the number of iterations towards a good enough plan was less or equal to 10.

6 Related Work

The notion of a socio-technical system, introduced almost 50 years ago [19], can be viewed from two different complementary perspectives: a social sciences [19, 43, 50] and a design or engineering sciences [46, 47]. Social sciences study psychological, managerial, and organizational aspects of a socio-technical phenomenon. For example, they are interested in relationships inside workgroups, roles, supervision, motivation and the like. Differently, the engineering perspective focuses on the design of socio-technical systems given sets of requirements, as well as their system properties. The latter perspective is the one we adopt in our work, like most of the research in software and requirements engineering. The basic ingredients of this perspective can be found in Simon's seminal vision of a Science of Design [46], where alternative designs are derived rigorously from their requirements (goals) and are evaluated in accordance with criteria that measure their effectiveness. The methodological differences in research and practice between the two perspectives are contrasted elegantly in [42].

Goal-based requirements modelling for socio-technical systems and organizations has been a topic of considerable research interest during the last fifteen years [49]. A number of goal-oriented approaches for requirements representation and reasoning were introduced, e.g. KAOS [15]. The i^* modelling framework [53] and an associated requirements analysis process, i.e. Tropos [7], are based on the intentional concepts of an actor, a goal and a social dependency, and support modelling and analysis activities during the requirements and design phases.

The field of AI planning has found a number of applications (robotics, process planning, autonomous agents, etc.). There are two basic approaches to the solution of planning problems [52]. One is graph-based planning algorithms in which a compact structure, called Planning Graph, is constructed and analyzed. In the other approach the planning problem is transformed into a SAT problem and a SAT solver is used.

There exist several ways to represent the elements of a classical planning problem, i.e. the initial state of the

world, the goal of the planning problem (i.e. or the desired state of the world), and the possible actions system actors can perform. PDDL (Planning Domain Definition Language) is the widely used specification language proposed in [27]. In the implementation of our approach, we use PDDL of the version 2.2 [18], which supports, among others, derived predicates and timed initial literals.

A few works relate planning techniques with information systems requirements analysis and design [2, 22, 13]. For example, one of the early proposals [2] describes a program called ASAP (Automated Specifier And Planner), which automates a part of the domain-specific software specification process. ASAP supports a designer in selecting methods for achieving user goals, discovering plans that result in undesirable outcomes, and finding methods for preventing such outcomes. The disadvantage of the approach is that a designer still performs a lot of work manually when determining the combination of goals and prohibited situations appropriate for a given application, defining possible start-up conditions and providing many other domain-specific expert knowledge.

Castillo et al. [13] present an AI planning application to assist an expert in designing control programs in the field of Automated Manufacturing. The system they have built integrates POCL, hierarchical and conditional planning techniques (see [13, 40] for references). The authors consider standard planning approaches to be not appropriate with no ready-to-use tools for the real world, whereas in our paper the opposite point of view is advocated. An application of the planning approach to the design of secure systems is proposed by Gans et al. [22]. Their work is based on i^* modelling approach [53] and ConGolog [28], a logic-based planning language. However, the authors focus more on representing/modelling trust in social networks, than on automating the design, and do not go far in explaining how they exploit the planning formalism.

Recently, the problem of evaluating and comparing requirements models has received a lot of attention, and there are several proposals in the literature which address the problem from different, often complementary perspectives. For example, a number of works [14, 29, 51] analyze the contribution links between goals and non-functional requirements in order to compare the alternative choices with respect to such non-functional requirements as security, efficiency, user-friendliness, etc. In [21] a problem of defining quantitative evaluation metrics for evaluating i^* models is discussed, with predictability of model elements as an example of the measured property. In AGORA requirements elicitation framework [31], each stakeholder has a preference matrix for each of the goals, which contains not only his preference value for the goal, but also his estimation of the preference values for this goal for other stakeholders. These matrices can aid a designer in selecting and adopting a goal from various alternatives by identifying and analyzing goal conflicts. Yet another i^*-related proposal [48] uses coupling metrics to assess the degree of dependencies between a system and its users. A closely related area is that of the value-based software engineering [6], which focuses on the definition and application of value-driven methods aimed at supporting the search for an optimal engineering solution.

7 Discussion and Future Work

We have proposed a systematic, tool-supported process which aim is to support a designer in exploring and evaluating alternative configurations of socio-technical system delegations. This is accomplishing through the use of AI planning techniques to construct design alternatives, and a set of evaluation criteria to assess and compare the available options.

In this paper, we have presented the structured process supporting the proposed approach, the planning formalization and the discussion on the concrete set of evaluation criteria. Moreover, we have reported on the evaluation of the proposal through several case studies (one of which is presented in detail here), as well as a series of scalability experiments.

What concerns the possible future work directions, we are interested in addressing the following problems:

- Adopt novel planning techniques that go beyond those we adopted and used in our work; for example, use concepts from ConGolog [28] or planning with preferences [5].
- As discussed in Section 4.1, incorporate into the framework an existing approache to quantitative and/or qualitative reasoning on non-functional system requirements, which will allow evaluating and comparing alternatives in terms of their impact on the satisfaction of NFR.
- Further explore criticality evaluation criteria, that is, define the criticality measures which characterize how difficult it is to repair a plan when an actor is removed from it; study the complexity issues related to these measures.
- Apply the proposed approach to the problem of system re-configuration at runtime. In [9] we have presented preliminary results along this direction.
- Work further on the evaluation of the approach using the real-life medium and large scale case studies, and conduct more scalability experiments.
- Continue the development of the tool [44] (see Section 3.5 for the details) to better support the proposed process.

Acknowledgements We thank the anonymous reviewers for the valuable comments and suggestions. This work has been partially funded by EU Commission through the SERENITY project, by MEnSA-PRIN project, and by the Provincial Authority of Trentino through the STAMPS project.

References

1. S. Albers, S. Eilts, E. Even-Dar, Y. Mansour, and L. Roditty. On Nash Equilibria for a Network Creation Game. In *SODA'06*, pages 89–98, 2006.
2. J. S. Anderson and S. Fickas. A Proposed Perspective Shift: Viewing Specification Design as a Planning Problem. In *IWSSD'89*, pages 177–184, 1989.
3. E. Anshelevich, A. Dasgupta, E. Tardos, and T. Wexler. Near-Optimal Network Design with Selfish Agents. In *STOC'03*, pages 511–520, 2003.
4. Y. Asnar, V. Bryl, and P. Giorgini. Using risk analysis to evaluate design alternatives. In *AOSE'06 Post-proceedings*, volume 4405, pages 140–155. Springer, 2006.
5. J. Baier, F. Bacchus, and S. McIlraith. A Heuristic Search Approach to Planning with Temporally Extended Preferences. In *IJCAI'07*, pages 1808–1815, 2007.
6. S. Biffl, A. Aurum, B. Boehm, H. Erdogmus, and P. Grünbacher. *Value-Based Software Engineering*. Springer-Verlag New York, Inc., 2005.
7. P. Bresciani, P. Giorgini, F. Giunchiglia, J. Mylopoulos, and A. Perini. TROPOS: An Agent-Oriented Software Development Methodology. *JAAMAS*, 8(3):203–236, 2004.
8. P. Bresciani, P. Giorgini, H. Mouratidis, and G. Manson. Multi-agent Systems and Security Requirements Analysis. *Software Engineering for Multi-Agent Systems II*, pages 35–48, 2004.
9. V. Bryl and P. Giorgini. Self-Configuring Socio-Technical Systems: Redesign at Runtime. In *SOAS'06*, 2006.
10. V. Bryl, P. Giorgini, and J. Mylopoulos. Designing Cooperative IS: Exploring and Evaluating Alternatives. In *CoopIS'06*, pages 533–550, 2006.
11. V. Bryl, P. Giorgini, and J. Mylopoulos. Supporting Requirements Analysis in Tropos: a Planning-Based Approach. In *PRIMA'07*, 2007.
12. V. Bryl, F. Massacci, J. Mylopoulos, and N. Zannone. Designing Security Requirements Models Through Planning. In *CAiSE'06*, pages 33–47, 2006.
13. L. Castillo, J. Fdez-Olivares, and A. Gonzalez. Integrating Hierarchical and Conditional Planning Techniques into a Software Design Process for Automated Manufacturing. In *ICAPS'03*, pages 28–39, 2003.
14. L. K. Chung, B. A. Nixon, E. Yu, and J. Mylopoulos. *Non-Functional Requirements in Software Engineering*. Kluwer Publishing, 2000.
15. A. Dardenne, A. van Lamsweerde, and S. Fickas. Goal-directed Requirements Acquisition. *Science of Computer Programming*, 20:3–50, 1993.
16. R. K. Dash, N. R. Jennings, and D. C. Parkes. Computational Mechanism Design: A Call to Arms. *IEEE Intelligent Systems*, 18(6):40–47, 2003.
17. M. Davidson and M. Garagnani. Pre-processing Planning Domains Containing Language Axioms. In *PlanSIG'02*, pages 23–34, 2002.
18. 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.
19. F. Emery. Characteristics of socio-technical systems. *London: Tavistock*, 1959.
20. M. Fox and D. Long. PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. *JAIR*, 20:61–124, 2003.
21. X. Franch. On the Quantitative Analysis of Agent-Oriented Models. In *CAiSE'06*, pages 495–509, 2006.
22. G. Gans, M. Jarke, S. Kethers, and G. Lakemeyer. Modeling the Impact of Trust and Distrust in Agent Networks. In *AOIS'01*, pages 45–58, 2001.
23. M. Garagnani. A Correct Algorithm for Efficient Planning with Preprocessed Domain Axioms. *Research and Development in Intelligent Systems XVII*, pages 363–374, 2000.
24. M. Garzetti, P. Giorgini, J. Mylopoulos, and F. Sannicolò. Applying Tropos Methodology to a real case study: Complexity and Criticality Analysis. In *WOA'02*, pages 7–13, 2002.
25. B. C. Gazen and C. A. Knoblock. Combining the Expressivity of UCPOP with the Efficiency of Graphplan. In *ECP'97*, pages 221–233, 1997.
26. A. Gerevini and D. Long. Plan Constraints and Preferences in PDDL3. Technical Report RT 2005-08-47, University of Brescia, Italy, 2005.
27. M. Ghallab, A. Howe, C. Knoblock, D. McDermott, A. Ram, M. Veloso, D. Weld, and D. Wilkins. PDDL – The Planning Domain Definition Language. In *AIPS'98*, 1998.
28. G. D. Giacomo, Y. Lesperance, and H. J. Levesque. ConGolog, a Concurrent Programming Language Based on the Situation Calculus. *Artificial Intelligence*, 121(1-2):109–169, 2000.
29. P. Giorgini, J. Mylopoulos, E. Nicchiarelli, and R. Sebastiani. Reasoning with Goal Models. In *ER'02*, pages 167–181, 2002.
30. D. Gross and E. S. K. Yu. From Non-Functional Requirements to Design through Patterns. *Requirements Engineering*, 6(1):18–36, 2001.
31. H. Kaiya, D. Shinbara, J. Kawano, and M. Saeki. Improving the Detection of Requirements Discordances Among Stakeholders. *Requirements Engineering*, 10(4):289–303, 2005.
32. J. Koehler, B. Nebel, J. Hoffmann, and Y. Dimopoulos. Extending Planning Graphs to an ADL Subset. In *ECP'97*, pages 273–285, 1997.
33. E. Koutsoupias and C. Papadimitriou. Worst-Case Equilibria. In *STACS'99*.
34. K. Lai, M. Feldman, I. Stoica, and J. Chuang. Incentives for Cooperation in Peer-to-Peer Networks. In *Workshop on Economics of Peer-to-Peer Systems*, 2003.
35. E. Letier and A. van Lamsweerde. Reasoning about Partial Goal Satisfaction for Requirements and Design Engineering. *SIGSOFT Softw. Eng. Notes*, 29(6):53–62, 2004.
36. LPG Homepage. LPG-td Planner. http://zeus.ing.unibs.it/lpg/.
37. T. Moscibroda, S. Schmid, and R. Wattenhofer. On the Topologies Formed by Selfish Peers. In *PODC'06*, pages 133–142, 2006.
38. M. E. J. Newman. The Structure and Function of Complex Networks. *SIAM Review*, 45(2):167–256, 2003.
39. M. J. Osborne and A. Rubinstein. *A Course in Game Theory*. MIT Press, 1994.
40. J. Peer. Web Service Composition as AI Planning – a Survey. Technical report, University of St. Gallen, 2005.

41. A. Polleres. *Advances in Answer Set Planning*. PhD thesis, Vienna University of Technology, Austria, 2003.
42. C. Potts and W. C. Newstetter. Naturalistic inquiry and requirements engineering: Reconciling their theoretical foundations. In *RE'97*, page 118, 1997.
43. G. Ropohl. Philosophy of socio-technical systems. In *In Society for Philosophy and Technology 4(3)*, 1999.
44. Security and Dependability Tropos Tool. http://sesa.dit.unitn.it/sistar_tool/.
45. Serenity: System Engineering for Security and Dependability. Deliverable A1.D2.1. Security and Privacy Requirements at Organizational Level. http://www.serenity-forum.org/Work-package-1-2.html.
46. H. A. Simon. *The Science of the Artificial*. MIT Press, 1969.
47. I. Sommerville. *Software engineering (7th ed.)*. Addison-Wesley, 2004.
48. A. G. Sutcliffe and S. Minocha. Linking Business Modelling to Socio-technical System Design. In *CAiSE'99*, pages 73–87, 1999.
49. A. van Lamsweerde. Requirements Engineering in the Year 00: a Research Perspective. In *ICSE'00*, pages 5–19, 2000.
50. G. H. Walker, N. A. Stanton, P. M. Salmon, and D. P. Jenkins. A review of sociotechnical systems theory: a classic concept for new command and control paradigms. *Theoretical Issues in Ergonomics Science*, 2008.
51. M. Weiss, D. Amyot, and G. Mussbacher. Formalizing Architectural Patterns with the Goal-oriented Requirement Language. In *VikingPLoP'06*, 2006.
52. D. S. Weld. Recent Advances in AI Planning. *AI Magazine*, 20(2):93–123, 1999.
53. E. S.-K. Yu. *Modelling Strategic Relationships for Process Reengineering*. PhD thesis, University of Toronto, 1996.
54. N. Zannone. *A Requirements Engineering Methodology for Trust, Security, and Privacy*. PhD thesis, University of Trento, 2006.