REQUIREMENTS ANALYSIS FOR SOCIO-TECHNICAL SYSTEMS: EXPLORING AND EVALUATING ALTERNATIVES.

Volha Bryl, Paolo Giorgini and John Mylopoulos

February 2006

Technical Report # DIT-06-006

Requirements Analysis for Socio-technical Systems: Exploring and Evaluating Alternatives

Volha Bryl
University of Trento, Italy
bryl@dit.unitn.it

Paolo Giorgini
University of Trento, Italy
paolo.giorgini@unitn.it

John Mylopoulos
University of Toronto, Canada
University of Trento, Italy
jm@cs.toronto.edu

Abstract

Early requirements analysis focuses on stakeholders and their goals, and explores alternative ways of fulfilling them through networks of actor delegations. The i modeling framework is a popular way of modeling and analyzing early requirements. This paper frames the problem of designing actor dependency networks as a multi-agent planning problem and adopts off-the-shelf planners to offer a tool that generate alternative actor dependency networks, and evaluate them in terms of metrics derived from Game Theory literature. The paper presents in detail how planning can generate alternative networks, also describes the tool (P-Tool) that supports the planning process and generates alternatives. As well, we offer preliminary experimental results on the scalability of the approach.*

1. Introduction

The last fifteen years have seen an increased emphasis on an early phase in Requirements Engineering (RE) when the focus is on stakeholders and their goals, rather than the system-to-be. *i** [20] is a modeling framework that supports modeling and analysis during this phase. According to *i** and an associated requirements analysis process (Tropos [2]), one begins requirements acquisition by identifying stakeholders ("actors") and their goals. These goals are decomposed and delegated to other actors, thereby creating networks of delegations. The process ends when all initial ("root") goals can be fulfilled if all actors deliver on their delegations. One or more of these actors represent the system-to-be. The functional requirements for the system-to-be are determined by all delegations to these system actors.

Exploring the space of alternative actor dependency networks is a difficult design task. This is so because such networks represent complex socio-technical systems where organizational, human and system actors depend on each other to fulfill root-level goals. Moreover, there are no generic criteria to guide the design process by determining whether a solution is good-enough, or even optimal. Our long-term objective is to develop such criteria and use them through tools and systematic design processes.

The purpose of this paper is to propose a framework for the automatic selection and evaluation of design alternatives. The framework supports both the generation and evaluation of alternatives. Specifically, the framework adopts multi-agent planning techniques and uses off-the-shelf planning tools. Alternatives are evaluated with respect to individual interests of system actors (i.e. their own goals). Ideas from Game Theory [14] are used to determine whether an alternative is an equilibrium. In particular, an alternative is in equilibrium if no actor can do better with respect to its own goals by adopting a different strategy for delegating and accepting delegations. When combined together, these two steps support the designer/requirements engineer in selecting alternatives that are in equilibrium with respect to the local strategies of each actor. An early version of this idea is used in [3] to propose a framework to generate alternative designs for secure systems. This paper goes further by describing a prototype tool that generates alternatives, presents some experimental results, and also proposes evaluation techniques for alternatives based on game-theoretic notions.

The process of the best alternative selection consists of the following steps:

1. Identify system and human actors, goals and their properties. Define goal decompositions and dependency relationships among actors.
2. For each actor identify criteria to evaluate alternatives.
3. Automatically explore the space of alternatives "on the upper level" to identify assignments of coarse-grained goals to actors.

4. Separately for each actor, automatically explore the alternative ways to satisfy the goals the actor was assigned at step 3. According to above identified evaluation criteria, select ”the best” alternative for each actor. During this step, alternative refinements of coarse-grained goals and delegation dependencies among actors are explored.

5. Evaluate the combined solution consisting of alternatives identified at step 4. In case it does not satisfy one or several system actors (e.g. they are overloaded with respect to others), return to step 4 to search for another alternative.

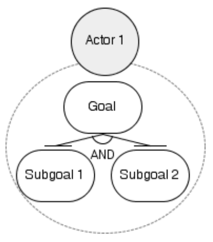
Ideally, the process stops after a number of iterations when the socio-technical structure is optimized enough to comply with the individual interests of the system actors. If no satisfactory alternatives can be generated at some step, the designer should return to steps 1 or 2, and revise either the initial structure, or the evaluation criteria.

Figure 1a presents a simple example of the problem. *Actor 1* has to achieve a *Goal*, which can be refined into two subgoals *Subgoal 1* and *Subgoal 2*. The actor can decide to achieve the goal by itself or delegate it to *Actor 2*. In both cases, there are a number of alternative ways that can be adopted. So, for instance, *Actor 1* can decide to delegate to *Actor 2* the whole *Goal* (Figure 1b), or part of it (Figure 1c). Shaded goal in the circle of an actor means that the goal is the responsibility of this actor. Even for this primitive example, exploring all the alternatives is quite tedious, and a support for alternative generation and evaluation would be beneficial.

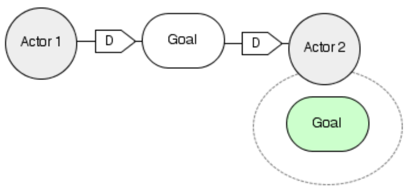
The rest of the paper is structured as follows. In the next Section we describe the example we use through the paper to describe our framework. In Section 3 the issue of alternative generation and evaluation is detailed. Section 4 describes the P-Tool, an implemented prototype tool to support the exploration of alternatives, and reports some experimental results. Finally, in Section 5 we describe the related work, and discuss conclusions and future work directions in Section 6.

2. SDS System example

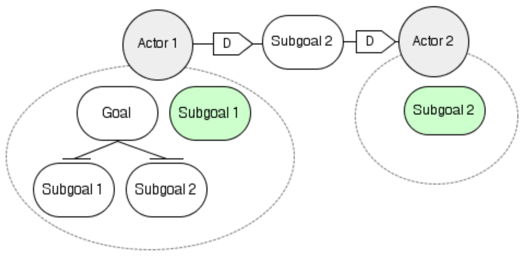
Let us consider a small software development company, which typical projects are medium-scale web based information systems (like, e.g. online library catalog, or travel agency home page with online trip booking, etc.). Within the company there are three teams of developers each focused on its area: GUI development, web design, and database support. Each team can develop subcomponents and/or consult other teams on questions related to their expertise. A manager is supposed to divide the project into meaningful parts and perform the assignment of goals to achieve to the development teams. Note, that a manager can assess how the project goals are refined and what skills it requires to satisfy each subgoal only on the coarse-grained level.



(a) Sample problem



(b) 1st alternative



(c) 2nd alternative

Figure 1. Sample problem and two alternative solutions

The company has decided to use a software development support system (SDS system or supporting system in the following), which will facilitate and report communication among actors, archive a library of reusable components, organize the search for such components, store and provide the information specific to the project under development (e.g. contain a glossary of domain specific terms, store domain specific classifications, etc.). Communication between manager and members of the development teams is supposed to be carried out only through the supporting system. Teams can communicate with each other in two cases: when one team wants to redirect a subgoal which requires the development skills these team does not possess to another team, and when one team needs to consult another one. The first type of communication is possible only through the supporting system, while the communication on consultancy can be done both through the supporting system and e-mail (or even personal communication).

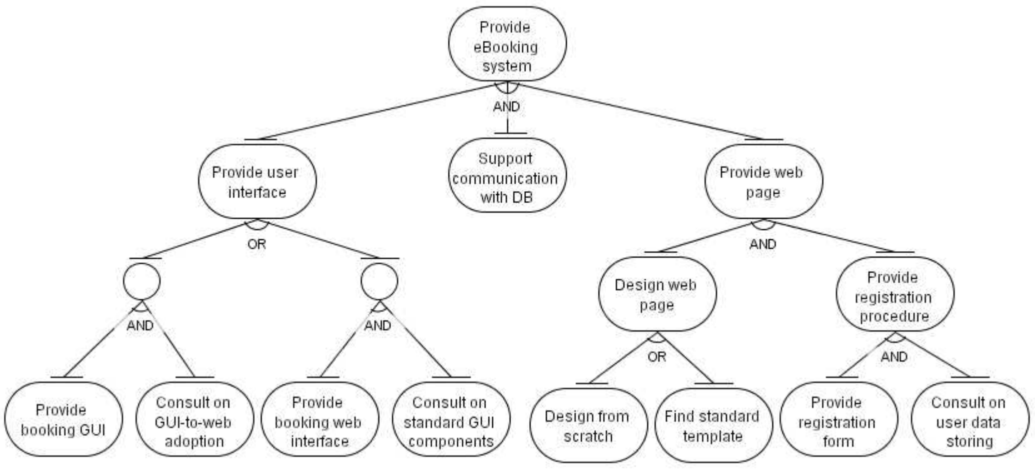

Provide eBooking system
AND
Provide user interface
Support communication with DB
Provide web page
OR
AND
AND
AND
Design web page
Provide registration procedure
OR
AND
Provide booking GUI
Consult on GUI-to-web adoption
Provide booking web interface
Consult on standard GUI components
Design from scratch
Find standard template
Provide registration form
Consult on user data storing

Figure 2. Goal tree for eBooking project

To analyze the above described socio-technical system let us consider a typical project it might deal with: web based eBooking system for the travelling agency. As it is represented in Figure 2, the high-level goal *provide eBooking system* is refined into three subgoals: *provide user interface*, *support communication with DB* and *provide web page*. In order to fulfil the high-level goal, all three subgoals should be satisfied. Two subgoals are further refined, e.g. *provide eBooking system* subgoal can be reached in two alternative ways: by developing eBooking GUI and consulting web designers to adopt it for web environment, or by developing web booking interface together with consulting GUI team on which standard components to use.

The OR-decomposition of the subgoals *provide user interface* and *design web page* introduces alternative solutions for the development of the eBooking system. One of the alternatives to achieve *provide user interface* subgoal is depicted in Figure 3. The goal is decomposed by *GUI team* actor, which selects the lefttmost alternative among the two or-subgoals. The selected subgoal is further decomposed into two subgoals: *consult on standard GUI components* and *provide booking web interface*. The former is satisfied by *GUI team*, while the latter is delegated to *Web design team*.

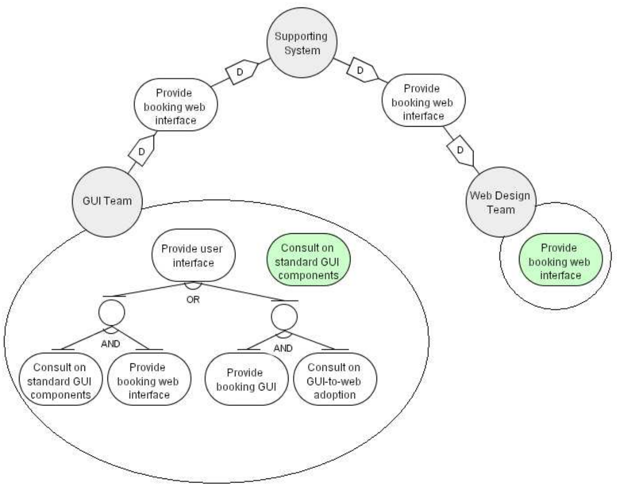

Supporting System
D
D
Provide booking web interface
Provide booking web interface
D
D
GUI Team
Web Design Team
Provide user interface
Consult on standard GUI components
Provide booking web interface
OR
AND
AND
Consult on standard GUI components
Provide booking web interface
Provide booking GUI
Consult on GUI-to-web adoption

Figure 3. An alternative way to achieve *provide user interface* **subgoal**

3. Exploring and evaluating alternatives

3.1. Formalization of the planning problem

It can be noticed that requirements – at least within the frameworks such as *i** [20], Tropos [2] and the like – 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 delegated goal, or further delegate it, thus creating another delegation relation in the network. Intuitively, these can be seen as actions that the designer/requirements engineer 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: selecting a suitable requirements structure corresponds to selecting a plan that satisfies the goals of human and software agents.

Thus, we have chosen the AI planning approach to support the designer/requirements engineer in the process of selecting the best alternative. The basic idea behind planning approach is to automatically determine the course of actions (i.e. a plan) needed to achieve a certain goal where an action is a transition rule from one state of the system to another [19, 15]. Actions are described in terms of preconditions and effects: if the precondition is true in the current state of the system, then the action is performed. As a consequence of an action, the system will be in a new state where the effect of the action is true.

Planning approach requires a specification language to represent the planning domain, i.e.

- the initial state of the system;
- the goal of the planning problem (i.e. the desired final state of the system);
- the description of actions;
- the axioms of background theory.

Once the domain is described, the solution to the planning problem is the (not necessarily optimal) sequence of actions that allows the system to reach the desired state from the initial state.

To describe the initial state of the system, actors' and goal properties, and social relations among actors should be specified. We propose to represent initial state in terms of predicates that correspond to

- the possible ways of goal decomposition;
- actors' capabilities and desires to achieve a goal;
- possible delegation relations among actors.

The desired state of the system (or the goal of the planning problem) is described through the conjunction of predicates derived from the description of actors' desires in the initial state. Essentially, for each desired goal a predicate is added to the goal of the planning problem.

Different types of logic could be applied for this purpose, e.g. first order logic is often used to describe the planning domain with conjunctions of literals specifying the states of the system. In table 1 predicates used to describe the requirements engineering domain are introduced. Predicates take variables of three types: actors, goals and goal types. To typify goals, *type* predicate is used. Actor capabilities are described with *can_decompose* and *can_decompose_gt* predicates, which mean that an actor has enough capabilities to satisfy a specific goal or any goal of a specific type, accordingly. Social dependencies among actors are reflected by *can_depend_on* and *can_depend_on_gt* predicates, which means that one actor can delegate to another actor the fulfilment of any goal or, in the latter case, any goal of a specific type. Predefined ways of goal refinement are represented using *decomposition* predicates, while with *can_decompose_gt* the scope of each actor can be represented: an actor can refine, or knows how to refine, only goals within his scope. Initial actors' desires are represented

Goal Properties
type(g : goal, gt : gtype)
subtype(child : gtype, parent : gtype)
and_decomposition$_n$(g : goal, g$_1$: goal, . . . , g$_n$: goal)
or_decomposition$_n$(g : goal, g$_1$: goal, . . . , g$_n$: goal)
satisfied(g : goal)
Actor Properties
can_satisfy(a : actor, g : goal)
can_satisfy_gt(a : actor, gt : gtype)
can_decompose_gt(a : actor, gt : gtype)
wants(a : actor, g : goal)
Actor Relations
can_depend_on(a : actor, b : actor)
can_depend_on_gt(a : actor, b : actor, gt : gtype)

Table 1. Primitive predicates

```
type (ConsultOnGUIToWebAdoption, tWDConsult)
subtype (tWDConsult, tConsult)
can_depend_on_gt (GUITeam, WDTeam, tConsult)
can_depend_on_gt (WDTeam, GUITeam, tConsult)

type (ProvideBookingWebInterface, tWDDevel)
type (DesignFromScratch, tWDDevel)
type (ProvideRegistrationForm, tWDDevel)
can_satisfy_gt (WDTeam, tWDDevel)

type (ProvideEBookingSystem, tManagScope)
can_decompose_gt (Manager, tManagScope)
```

Figure 4. Predicates for SDS System example

with *wants* predicate. When the goal is fulfilled *satisfied* predicated becomes true for it.

In figure 4, a part of SDS System example formalization is presented. The goal types *tConsult*, *tWDDevel* and *ManagScope* are used.

In *i**/Tropos approach, when drawing the model of a system, the designer/requirements engineer assigns goals to actors, defines delegations of goals from one actor to another, and identifies appropriate goal refinements among the predefined alternative refinements. Thus, the following actions will be used by a planner to find a way to fulfill the goals of the system actors.

Goal satisfaction. An actor can satisfy a goal only if the achievement of the goal is among his desires and he can actually satisfy it. The effect of this action is the fulfillment of the goal.

Goal delegation. An actor may have not enough capabilities to achieve his goals by himself, and so he has to delegate their satisfaction to other actors. This passage of responsibilities is performed only if the delegator wants a goal to be achieved and can depend on the delegatee to achieve it. The effect of this action is that the delegator does not worry any more about the satisfaction of the goal, while the delegatee takes the responsibility for the fulfillment of the goal and so it becomes his own desire to achieve it. The delegator does not care how the delegatee satisfies the goal (e.g. by his own capabilities or by further delegation), it is up to the delegatee to decide it.

Goal decomposition/refinement. As in different goal-oriented modeling frameworks (e.g. as in Tropos and KAOS [5]) two types of goal refinement are supported: OR-decomposition, which suggests the list of alternative ways to satisfy the goal, and AND-decomposition, which refines the goals into subgoals which all are to be satisfied in order to satisfy the initial goal. An actor can decompose a goal only if he wants it to be satisfied, and only in the way which is predefined in the initial state of the system. The effect of decomposition is that the actor who refines the goal focuses on the fulfillment of subgoals instead of the initial goal. It is assumed that different actors can decompose the same goal in different ways.

In addition to actions, axioms of the planning domain can be defined. These are rules that hold in every state of the system and are used to complete the description of the current state. For example, to propagate goal properties along goal refinement the following axiom is used: a goal is satisfied if all its and-subgoals or at least one of the or-subgoals are satisfied.

3.2. Evaluation procedure

To complete the requirements analysis process, the alternative designs generated by the planner should be evaluated, amended and approved by the designer. The tricky point here is the solution evaluation which can be complex enough even for experienced designers with considerable domain expertise. Alternative requirements structures 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 individual actors. The optimality of a solution in the global sense could be assessed with respect to the following.

- *Length of the obtained plan.* The number of actions in the obtained plan is often the criteria for the planner itself to prefer one solution to another. Thus, it can be assumed that the obtained plan is already (locally) optimal in the sense of the length minimization.

- *Overall plan cost.* This is closely related with the idea of plan metrics introduced in PDDL 2.1 [8]. Plan metrics specify the basis on which a plan is evaluated for a particular problem (e.g. action costs or duration), and are usually numerical expressions to be minimized or maximized. However, the complexity of the problem of optimizing a solution with respect to the defined metrics is very high and the feature is still poorly supported by the available planning tools [8].

- *Degree of satisfaction of non-functional requirements.* E.g. in [12], a set of rules is proposed to identify application-specific parameters and functions to quantify impacts of different explored alternatives on non-functional goals (e.g. security, performance, usability) satisfaction.

Local evaluation of the obtained plan is a much more complex task. Indeed, a challenging characteristic of socio-technical IT system design is that human agents should be taken into account. They can be seen as players in a game theoretic sense as they are self-interested and rational. This means they want to minimize the load imposed personally on them, i.e. they want to reduce the number and the complexity of actions they are involved in. In a certain sense non-human agents, i.e. system components, 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 whether to satisfy a goal himself or to pass it further to another system actor. Strategies are based on the player's capabilities and his relations (e.g. subordination, friendship, or trust) with other human and artificial agents in the system.

The substantial difficulty in applying game theoretic ideas to our problem is that all actors of a socio-technical system should work as a solid mechanism satisfying the overall organizational goal. Differently from classical non-cooperative game theory, where all players choose their strategies independently and simultaneously before the game, in our problem actors' 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. Thus, to satisfy the system goals it will be necessary to impose some additional load (to compensate the one this player tries to avoid) on some other actors – and it might happen that they will not be satisfied with their new utilities, 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-plan again. The process stops when a (sub)optimal requirements alternative option is found. In our framework the following "replan-towards-optimality" procedure is used.

First, 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 costs are defined:

- cs_{ik} is the cost for the actor a_i of satisfying the goal

Action	Cost	Actors and Goals
Satisfy	3	goals of type *tConsult* for *WDTeam, GUITeam* and *DBTeam*; goal *FindStandardTemplate* for *SupportingSystem*
	4	goals of type *tWDDevel* for *WDTeam*, goal *ProvideBookingGUI* for *GUITeam*; goal *SupprtDBCommunication* for *DBTeam*
Delegate	1	*SupportingSystem*; delegations between *WDTeam, GUITeam* and *DBTeam*
	2	all other actors and goals
Refine	2	all actors and goals

Table 2. Costs for the SDS System example

g_k;

- cr_{ik} is the cost for the actor a_i of refining the goal g_k;
- cd_{ijk} is the cost for the actor a_i of delegating a goal g_k to the actor b_j.

For example, the costs of actions for actors from the SDS System example are defined in Table 2.

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

$$c\left(P, a_i\right) = \sum_{delegate(a_i, b_j, g_k) \in P} cd_{ijk} +$$

$$\sum_{decompose_l(a_i, g_k, g_{k1}, \ldots, g_{kl}) \in P} cr_{ik} + \sum_{satisfy(a_i, g_k) \in P} cs_{ik},$$

where $decompose_l(a_i, g_k, g_{k1}, ..., g_{kl})$ stands for the decomposition of g_k into l subgoals $g_{k1}, ..., g_{kl}$.

If P is the alternative depicted in Figure 3, then $c\left(P, GUITeam\right) = 6$, $c\left(P, WDTeam\right) = 6$ and $c\left(P, SupportingSystem\right) = 2$.

Note, that in our framework we do not use the notion of utility, which is an important game theory construct. This is done mainly for the simplicity reasons. The utility of an alternative P for the actor a_i can be defined as the difference between maximum upper bound for the solution cost for actor a_i and $c\left(P, a_i\right)$. Basically, utility says how much an actor ”saves” with the alternative P being selected.

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 upper cost bound c_i^{up} is greater than $c\left(P, a_i\right)$ is willing to deviate from P.
- Actor a_i whose predefined upper bound $cdev_i^{up}$ on cost deviation is greater than $c\left(P, a_i\right) - avg_i(c\left(P, a_i\right))$ wants to deviate from P.

Finally, the evaluation procedure is the following.

- An alternative P is generated with the help of planner.
- Cost $c\left(P, a_i\right)$ is calculated for each a_i.
- Actor a_{min} is identified whose value of $c\left(.,.\right)$ is minimal among all actors who want to deviate from P.
- The first most expensive action d_{worst} is identified among actions of P in which a_{min} is involved.
- Negation of d_{worst} is added to the initial planning problem, and replanning is performed. If no plan can be found, the next d_{worst} is identified.

The process stops when an equilibruium-like solution is found, i.e. no actors are willing to deviate from it and the designer approves this solution. The designer remains in the process all the time, and can stop the iterations whenever he thinks the satisficing alternative is generated.

This evaluation procedure is used at the following steps of the selection of the best alternative, defined in the Introduction.

- At step 3, while selecting the best assignments of coarse-grained goals to actors.
- At step 4, separately for each actor, when exploring the ways to satisfy the goals the actor was assigned.
- At step 5, when evaluating the combined solution consisting of alternatives identified at step 4. Here the replanning is performed only for the alternative to which d_{worst} belongs to.

4. P-Tool and experiments

4.1. Choosing the planner

One important step we have performed during the implementation of the proposed framework, is choosing the ”right planner” among off-the-shelf tools available. In the last years many planners have been proposed [15]. In order to choose one of them the following requirements are considered:

- The planner should not produce redundant plans. Under non-redundant plan we mean that, by deleting an arbitrary action of the plan, the resulting plan is no more a ”valid” plan (i.e. it does not allow to reach the desired state from the initial state).
- The planner should use PDDL (Planning Domain Definition Language) since it is becoming a ”standard” planning language and many research groups work on its implementation.

```
(: action Satisfies
  : parameters(?a − t_actor, ?g − t_goal
  : precondition (and
     (or(can_satisfy?a?g)
        (exists(?gt − t_gtype)(and(type?g?gt)
           (can_satisfy_gt?a?gt)))
     (wants?a?g)
  : effect (and
     (satisfied?g)
     (not(wants?a?g))))

(: derived
  (type?g − t_goal?parent − t_gtype)
  (exists(?child − t_gtype)
     (and(subtype?child?parent)(type?g?child))))
```

Figure 5. Domain description using PDDL

- The language should support a number of ”advanced” features (e.g. derived predicates) that are essential for implementing our planning domain, i.e. it should be at least PDDL 2.2. [6].

The first requirement is concerned with the optimality of the generated design decisions. We argue that it is not necessary to focus on the optimal design: human designers do not prove that their design is optimal, why should a system do it? Instead, in our framework the plan is required to be non-redundant, which guarantees at least the absence of alternative delegation paths since a plan does not contain any redundant actions.

We have compared a number of planners with respect to above requirements (see [3] for the details). Finally, we have chosen LPG-td [13], a fully automated system for solving planning problems, supporting PDDL 2.2 specification for implementing our planning domain.

Then, we have implemented our planning domain in PDDL 2.2. Figure 5 presents the specification of one action and one domain axiom in PDDL 2.2.

Figure 6 shows the plans generated by LPG-td for satisfying *provide user interface* and *provide web page* subgoals. The former plan is illustrated in Figure 3, the latter – in Figure 7.

Preliminary experiments were conducted to test the scalability of the approach. A very simple ”core” problem was considered, with three actors A, B and C and two goals, G_1 and G_2, which A wants to be achieved, and B and C can satisfy. Then ”additional” actors with the dependencies among them were added to the problem, but they did not interfere at all with ”core” subproblem. The idea was to check whether the search time of the plan to achieve G_1 and G_2 depends on the number of ”additional” actors and dependencies among them. The experiments showed that, at least with respect to this example, the approach is scalable. Basically, the search time for the problem with 10 and with 120 ”additional” actors is the same (less than one second), only the parsing time increases insignificantly. At the same time, search time for the plan with long delegation chains (more than 30 steps) is much greater (around 15 seconds). Of course, the scalability issue should be explored much more carefully (actually, this is one of our future work plans), but the above reported preliminary experiments have shown promising results.

```
(OR_DECOMPOSES GUITeam ProvideUI ProvideUI1 ProvideUI2)
(AND_DECOMPOSES GUITeam ProvideUI1
     ProvideBookingWebI ConsultStandGUI)
(SATISFIES GUITeam ConsultStandGUI)
(PASSES GUITeam SupportingSystem ProvideBookingWebI)
(PASSES SupportingSystem WDTeam ProvideBookingWebI)
(SATISFIES WDTeam ProvideBookingWebI)
```

(a) Provide user interface

```
(AND_DECOMPOSES WDTeam ProvideWebPage
     DesignWebPage ProvideRegistrProc)
(AND_DECOMPOSES WDTeam ProvideRegistrProc
     ProvideRegForm ConsultOnStoreUData)
(SATISFIES WDTeam ProvideRegForm)
(OR_DECOMPOSES WDTeam DesignWebPage
     DesignFromScratch FindStandardTemplate)
(SATISFIES WDTeam DesignFromScratch)
(PASSES WDTeam DBTeam ConsultOnStoreUData)
(SATISFIES DBTeam ConsultOnStoreUData)
```

(b) Provide web page

Figure 6. Plans for *ProvideUI* and *ProvideWebPage* subgoals

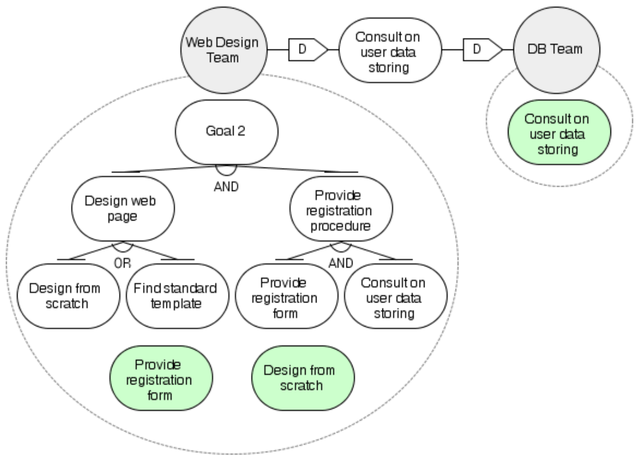
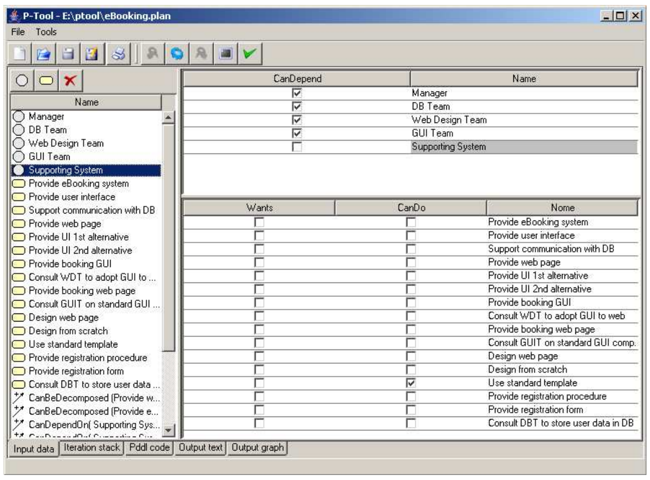

Web Design Team
D
Consult on user data storing
D
DB Team
Consult on user data storing
Goal 2
AND
Design web page
Provide registration procedure
OR
AND
Design from scratch
Find standard template
Provide registration form
Consult on user data storing
Provide registration form
Design from scratch

Figure 7. Diagram for the plan for *ProvideWebPage* subgoal

4.2. P-Tool

We have developed P-Tool, an implemented prototype to support the designer/requirements engineer in the process of exploring and evaluating alternatives. The tool has the interface for the input of actors, goals and their properties, which can be seen in Figure 8. LPG-td is built in the tool, and is used to generate requirements alternatives, and represents each solution graphically using *i** notation, see Figure 9 for an example.


P-Tool - E:\ptool\eBooking.plan

File Tools

Name
Manager
DB Team
Web Design Team
GUI Team
Supporting System
Provide eBooking system
Provide user interface
Support communication with DB
Provide web page
Provide UI 1st alternative
Provide UI 2nd alternative
Provide booking GUI
Consult WDT to adopt GUI to ...
Provide booking web page
Consult GUIT on standard GUI ...
Design web page
Design from scratch
Use standard template
Provide registration procedure
Provide registration form
Consult DBT to store user data ...
CanBeDecomposed (Provide w...
CanBeDecomposed (Provide e...
CanDependOn(Supporting Sys...

CanDepend	Name
☑	Manager
☑	DB Team
☑	Web Design Team
☑	GUI Team
☐	Supporting System

Wants	CanDo	Nome
☐	☐	Provide eBooking system
☐	☐	Provide user interface
☐	☐	Support communication with DB
☐	☐	Provide web page
☐	☐	Provide UI 1st alternative
☐	☐	Provide UI 2nd alternative
☐	☐	Provide booking GUI
☐	☐	Consult WDT to adopt GUI to web
☐	☐	Provide booking web page
☐	☐	Consult GUIT on standard GUI comp.
☐	☐	Design web page
☐	☐	Design from scratch
☐	☑	Use standard template
☐	☐	Provide registration procedure
☐	☐	Provide registration form
☐	☐	Consult DBT to store user data in DB

Input data | Iteration stack | Pddl code | Output text | Output graph

Figure 8. P-Tool: identifying actor properties

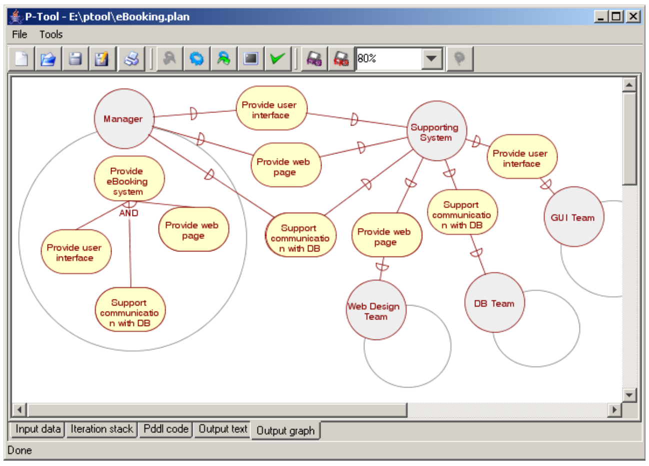


Figure 9. P-Tool: i* diagram for the generated alternative

In the following we will illustrate how the steps 3–5 of our approach (see Section 1) could be supported by the P-Tool. For the sake of simplicity we will leave out some details.

```
(AND_DECOMPOSES WDTeam ProvideWebPage
    DesignWebPage ProvideRegistrProc)
(AND_DECOMPOSES WDTeam ProvideRegForm
    ConsultOnStoreUData)
(SATISFIES WDTeam ProvideRegForm)
(PASSES WDTeam DBTeam ConsultOnStoreUData)
(SATISFIES DBTeam ConsultOnStoreUData)
(OR_DECOMPOSES WDTeam DesignWebPage
    DesignFromScratch FindStandardTemplate)
(PASSES WDTeam SupportingSystem FindStandardTemplate)
(SATISFIES SupportingSystem FindStandardTemplate)
```

Figure 10. New plan for *ProvideWebPage* subgoal

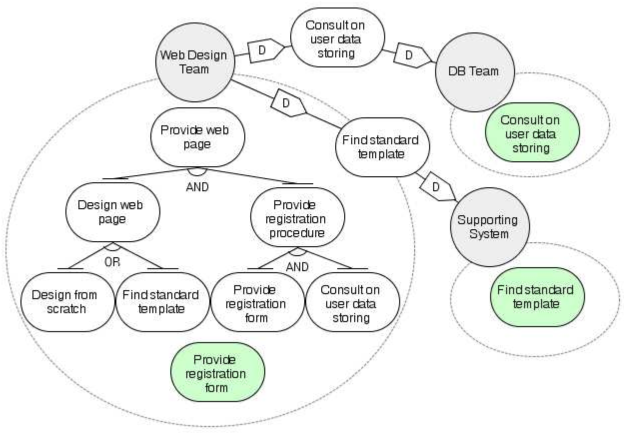


Figure 11. Diagram for new plan for *ProvideWebPage* subgoal

Step 3. First, the planning ”on the upper level” for *Manager* actor is performed. We will skip the process description. The resulting alternative can be seen in the screenshot in Figure 9. *Manager* decomposes *ProvideEBookingSystem* goal into *ProvideUI*, *ProvideWebPage* and *SupportDBCommunication* subgoals, and passes them through the *SupportingSystem* to *GUITeam*, *WDTeam* and *DBTeam*, respectively.

Step 4. We will illustrate this step with exploring alternatives for the subgoal *ProvideWebPage* assigned to *WDTeam* actor. Firstly, an alternative presented in Figure 6 is generated. For this alternative $c(P_1, WDTeam) = 2+2+4+2+4+2 = 16$, which does not satisfy *WDTeam* actor, so it tries to decrease the imposed load. According to the evaluation procedure described in Section 3.2, the action (SATISFIES WDTeam ProvideRegForm) is selected as d_{worst}. When this action is negated, the planner is not able to find a solution. Thus, the next d_{worst} is identified, which is (SATISFIES WDTeam DesignFromScratch). New alternative is generated, see Figures 10 and 11, for which

$c\,(P_2, WDTeam) = 2+2+4+2+2+1 = 13$. This last alternative is then fixed as it satisfies *WDTeam* actor.

Step 5. When partial plans are combined into the plan P and evaluated, it appears that $c\,(P, GUITeam) = 6$ and $c\,(P, WDTeam) = 13 + 6 = 19$. Actor *WDTeam* tries to deviate from the alternative P, and (SATISFIES WDTeam ProvideBookingWebI) of the plan depicted in Figure 6 is identify as d_{worst} and negated. By replanning we get an alternative presented in Figures 12 and 13, for which $c\,(P', GUITeam) = 2+2+4+2 = 10$ and $c\,(P', WDTeam) = 13 + 3 = 16$. This new alternative satisfies both *GUITeam* and *WDTeam* actors.

```
(OR_DECOMPOSES GUITeam ProvideUI ProvideUI1 ProvideUI2)
(AND_DECOMPOSES GUITeam ProvideUI1
    ProvideBookingWebI ConsultStandGUI)
(SATISFIES GUITeam ProvideBookingGUI)
(PASSES GUITeam WDTeam ConsultOnGUIToWebAdoption)
(SATISFIES WDTeam ConsultOnGUIToWebAdoption)
```

Figure 12. New plan for *ProvideUI* subgoal

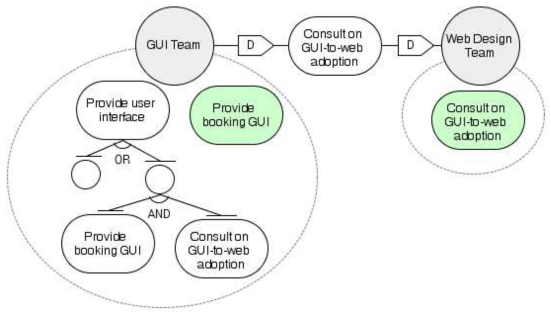

GUI Team
D
Consult on GUI-to-web adoption
D
Web Design Team
Provide user interface
Provide booking GUI
Consult on GUI-to-web adoption
OR
AND
Provide booking GUI
Consult on GUI-to-web adoption

Figure 13. Diagram for new plan for *ProvideUI* subgoal

5. Related work

Requirements engineering is considered to be a crucial part of software development process [18]. Careful elicitation and analysis of requirements help to develop a system that meets user's expectations, is trustful and robust. Modeling requirements to software systems and organizations in terms of goals and their interdependences has been a topic of considerable research interest during the last decades [18]. A number of goal-oriented approaches for requirements representation and reasoning were introduced, e.g. KAOS [5].

The field of AI planning has been intensively developing during the last decades, and has found a number of applications (robotics, process planning, autonomous agents, etc.). Planning approach recently has proved to be applicable in the field of automatic Web service composition [15]. There are two basic approaches to the solution of planning problems [19]. 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 system goal, or the desired state of the world, and the possible actions system actors can perform. The widely used, and to the certain extend standard representation is PDDL (Planning Domain Definition Language), the problem specification language proposed in [10]. Current PDDL version, PDDL 2.2 [6] used during the last International Planning Competition [11], supports many useful features, e.g. derived predicates and timed initial literals.

A few works can be found which relate planning techniques with software requirements analysis and design. In [1] a program called ASAP (Automated Specifier And Planner) is described, which automates a part of the domain-specific software specification process. ASAP assists the 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 the designer still performs a lot of work manually determining the combination of goals and prohibited situations appropriate for the given application, defining possible start-up conditions and providing many other domain-specific expert knowledge.

Castillo et al. [4] 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 [4, 15] for references). The authors consider standard planning approaches to be not appropriate with no ready-to-use tools for the real world, while in our paper the opposite point of view is advocated. Another recent application of the planning approach to requirements engineering for the secure systems is proposed by Gans et al. [9]. The work is based on *i** modeling approach [20] and ConGolog (see [15] for description and references), a logic-based planning language. However, the authors focus more on representing/modeling trust in social networks, than on the design automation, and do not go far in explaining how they exploit the planning formalism.

Game theory is an established discipline which deals with conflicts and cooperation among rational independent decision-makers, or players. The key concept in classical game theory is the notion of equilibrium [14] which defines the set of strategies, one for each player, which none of the 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.

Game theory is applied in various areas, especially in economics (modeling markets, auctions, etc.), corporate decision making, defense strategy, telecommunications networks and many others. Among the examples are the applications of game theory to so called network games (e.g. routing, bandwidth allocation, etc.), see [17] for references.

6. Conclusions

We have proposed a framework for automatically generating a space of alternative actor dependency networks that satisfy an initial set of stakeholder goals. The framework uses planning techniques to characterize the space of alternatives. A prototype tool with a built-in off-the-shelf planner is used to generate alternatives. These are evaluated in terms of criteria founded on game-theoretic notions.

This is clearly a first step towards making more systematic and tool-supported the process of designing actor dependency models for a given set of initial stakeholder goals. More needs to be done to ensure the scalability of the P-Tool. In particular, we'd like to include the use of heuristic (e.g., *A**-like [16]) techniques to reduce the space of alternatives under considering by filtering away early on alternatives that look bad. We'd also like to adopt proposals for better structuring actor dependency models. One such proposal [7] is to make *i** models "service-oriented" by encapsulating composite actors and allowing delegations to it only through a well-defined service interface. Such proposals reduce dramatically the number of possible solutions to a given multi-actor planning problem.

7. Acknowledgements

We thank Alfonso Gerevini and Alessandro Saetti for the support on the use of LPG-td planner. This work has been partially funded by EU Commission, through the SENSORIA and SERENITY projects, by the FIRB program of MIUR under the ASTRO project, and also by the Provincial Authority of Trentino, through the MOSTRO project.

References

[1] J. S. Anderson and S. Fickas. A proposed perspective shift: viewing specification design as a planning problem. In *IWSSD '89: 5th Int. workshop on Software specification and design*, pages 177–184, 1989.

[2] P. Bresciani, P. Giorgini, F. Giunchiglia, J. Mylopoulos, and A. Perini. TROPOS: An Agent-Oriented Software Development Methodology. 8(3):203–236, 2004.

[3] V. Bryl, F. Massacci, J. Mylopoulos, and N. Zannone. Designing secure systems through planning. In *CAiSE*, 2006. To appear.

[4] L. Castillo, J. Fdez-Olivares, and A. Gonzlez. Integrating hierarchical and conditional planning techniques into a software design process for automated manufacturing. In *ICAPS 2003, Workshop on Planning under Uncertainty and Incomplete Information*, pages 28–39, 2003.

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

[6] S. Edelkamp and J. Hoffmann. Pddl2.2: The language for the classical part of the 4th international planning competition. Technical Report 195, University of Freiburg, 2004.

[7] H. Estrada. Private communication.

[8] M. Fox and D. Long. Pddl2.1: An extension to pddl for expressing temporal planning domains. *J. Artif. Intell. Res. (JAIR)*, 20:61–124, 2003.

[9] G. Gans, M. Jarke, S. Kethers, and G. Lakemeyer. Modeling the impact of trust and distrust in agent networks. pages 45–58, 2001.

[10] M. Ghallab, A. Howe, C. Knoblock, D. McDermott, A. Ram, M. Veloso, D. Weld, and D. Wilkins. PDDL – The Planning Domain Definition Language. 1998.

[11] IPC-4 Homepage. International Planning Competition 2004. http://ls5-www.cs.uni-dortmund.de/ edelkamp/ipc-4/.

[12] 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.

[13] LPG Homepage. LPG-td Planner. http://zeus.ing.unibs.it/lpg/.

[14] M. J. Osborne and A. Rubinstein. *A Course in Game Theory*. MIT Press, 1994.

[15] J. Peer. Web Service Composition as AI Planning – a Survey. Technical report, University of St. Gallen, 2005.

[16] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, second edition, 2002.

[17] E. Tardos. Network games. In *Proceedings of the Annual ACM Symposium on Theory of Computing*, 2004.

[18] A. van Lamsweerde. Requirements engineering in the year 00: a research perspective. pages 5–19. ACM, 2000.

[19] D. S. Weld. Recent Advances in AI Planning. *AI Magazine*, 20(2):93–123, 1999.

[20] E. S.-K. Yu. *Modelling strategic relationships for process reengineering*. PhD thesis, University of Toronto, 1996.