

# Designing Security Requirements Models through Planning

Volha Bryl and Fabio Massacci and John Mylopoulos and Nicola Zannone<sup>1</sup>

**Abstract.** The quest for designing secure and trusted software has led to refined Software Engineering methodologies that rely on tools to support the design process. Automated reasoning mechanisms for requirements and software verification are by now a well-accepted part of the design process, and model driven architectures support the automation of the refinement process. We claim that we can further push the envelope towards the automatic exploration and selection among design alternatives and show that this is concretely possible for Secure Tropos, a requirements engineering methodology that addresses security and trust concerns. In Secure Tropos, a design consists of a network of actors (agents, positions or roles) with delegation/permission dependencies among them. Accordingly, the generation of design alternatives can be accomplished by a planner which is given as input a set of actors and goals and generates alternative multi-agent plans to fulfill all given goals. We validate our claim with a case study using a state-of-the-art planner.

## 1 Introduction

The design of secure and trusted software that meets stakeholder needs is an increasingly hot issue in Software Engineering (SE). This quest has led to refined Requirements Engineering (RE) and SE methodologies so that security concerns can be addressed during the early stages of software development (e.g. Secure Tropos vs *i\**/Tropos, UMLsec vs UML, etc.). Moreover, industrial software production processes have been tightened to reduce the number of existing bugs in operational software systems through code walkthroughs, security reviews etc. Further, the complexity of present software is such that all methodologies come with tools for automation support.

The tricky question in such a setting is what kind of automation? Almost fifty years ago the idea of actually deriving code directly from the specification (such as that advocated in [22]) started a large programme for deductive program synthesis,<sup>2</sup> that is still active now [5, 11, 25, 29]. However, proposed solutions are largely domain-specific, require considerable expertise on the part of their users, and in some cases do not actually guarantee that the synthesized program will meet all requirements stated up front [11].

Another approach is to facilitate the work of the designer by supporting tedious aspects of software development by automating the design refinement process. This approach underlies Model Driven Architectures (MDA) [27], which focuses on the (possibly auto-

matic) transformation from one system model to another. Tools supporting MDA exist and are used in the Rational Unified Process for software development in UML. Yet, the state-of-the-art is still not satisfactory [30].

Such approaches only cover part of the work of the designer. We advocate that there is another activity where the support of automation could be most beneficial [20]:

“Exploring alternative options is at the heart of the requirements and design processes.”

Indeed, in most SE methodologies the designer has tools to report and verify the final choices (be it goal models in KAOS, UML classes, or Java code), but not actually the possibility of automatically exploring design alternatives (i.e. the *potential choices* that the designer may adopt for the fulfillment of system actors’ objectives) and finding a satisfactory one. Conceptually, this automatic selection of alternatives is done in deductive program synthesis: theorem provers select appropriate axioms to establish the system goal. Instead, we claim that the automatic selection of alternatives should and indeed can be done during the very early stages of software development. After all, the automatic generation of alternatives is most beneficial and effective during these stages.

There are good reasons for this claim. Firstly, during early stages the design space is large, and a good choice can have significant impact on the whole development project. Supporting the selection of alternatives could lead to a more thorough analysis of better quality designs with respect to security and trust. Secondly, requirements models are by construction simpler and more abstract than implementation models (i.e. code). Therefore, techniques for automated reasoning about alternatives at the early stages of the development process may succeed where automated software synthesis failed.

Since our overall goal is to design a secure system we have singled out the Secure Tropos methodology [16] as the target for our work. Its primitive concepts include those of Tropos and *i\** [7], but also concepts that address security concerns, such as ownership, permission and trust. Further, the framework already supports the designer with automated reasoning tools for the verification of requirements as follows:

1. Graphical capture of the requirements for the organization and the system-to-be,
2. Formal verification of the functional and security requirements by
  - completion of the model drawn by the designer with axioms (a process hidden to the designer);
  - checking the model for the satisfaction of formal properties corresponding to specific security or design patterns.

<sup>1</sup> University of Trento, Italy, email: {bryl,massacci,jm,zannone}@dit.unitn.it

<sup>2</sup> A system goal together with a set of axioms are specified in a formal specification language. Then the system goal is proved from the axioms using a theorem prover. A program for achieving the goal is extracted from the proof of the theorem.

In this framework (as in many other similar RE and SE frameworks) the selection of the alternatives is left to the designer. We will show that we can do better.

Indeed, in Tropos (resp. Secure Tropos) requirements are conceived as networks of functional dependencies (resp. delegation of execution) among actors (organizational/human/software agents, positions and roles) for goals, tasks and resources. Every dependency (resp. delegation of execution) also involves two actors, where one actor depends on the other for the delivery of a resource, the fulfillment of a goal, or the execution of a task. Intuitively, these can be seen as *actions* that the designer has ascribed to the members of the organization and the system-to-be. As suggested by Gans et al. [14] the task of designing such networks can then be framed as a planning problem for multi-agent systems: selecting a suitable possible design corresponds to selecting a plan that satisfies the prescribed or described goals of human or system actors. Secure Tropos adds to the picture also the notion of delegation of permission and various notions of trust.

In this paper we show that it is possible to use an off-the-shelf planner to select among the potential dependencies the actual ones that will constitute the final choice of the requirements engineer. If a planner is already able to deliver good results then this looks a promising avenue for transferring the technique to complex industry-level case studies where a customized automated reasoning tool might be very handy. At the same time, if the problem is not trivial, not all planners will be able to deliver and indeed this turned out to be the case. The techniques we use are sufficiently powerful to cope with security requirements as well as functional requirements, but we concentrate here on their applicability to a security setting where an automated support for the selection of potentially conflicting alternatives is more urgent. The application of the same planning techniques to the overall software development phases can be found in [3].

In this work we have not focused on optimal designs: after all, human designers do not aim for optimality in their designs. As noted by Herbert Simon in his lecture on a “Science of Design” [31] what makes humans effective (in comparison to machines) is their ability to identify a satisficing design as opposed to an optimal one.

Of course, we assume that the designer remains in the loop: designs generated by the planner are suggestions to be refined, amended and approved by the designer. The planner is a(nother) support tool intended to facilitate the design process.

The rest of the paper is structured as follows. Section 2 explains Secure Tropos concepts and describes the requirements verification process. In Sections 3, 4 and 5 the planning approach to the system design is introduced and explained, while in Section 6 the implementation of our approach is presented. Finally, in Sections 7 and 8 a brief overview of related work is presented and conclusions are drawn.

## 2 Secure Tropos

Secure Tropos [16] is a RE methodology for modeling and analyzing functional and security requirements, extending the Tropos methodology [7]. This methodology is tailored to describe both the system-to-be and its organizational environment starting with early phases of the system development process. The main advantage of this approach is that one can capture not only the *what* or the *how*, but also the *why* a security mechanism should be included in the system design. In particular, Secure Tropos deals with business-level (as opposed to low-level) security requirements. The focus of such requirements includes, but is not limited to, how to build trust among different partners in a virtual organization and trust management. Al-

though their name does *not* mention security, they are generally regarded as part of the overall security framework.

Secure Tropos uses the concepts of actor, goal, task, resource and social relations for defining entitlements, capabilities and responsibilities of actors. An *actor* is an intentional entity that performs actions to achieve goals. A *goal* represents an objective of an actor. A *task* specifies a particular sequence of actions that should be executed for satisfying a goal. A *resource* represents a physical or an informational entity.

Actors’ desires, entitlements, capabilities and responsibilities are defined through social relations. In particular, Secure Tropos supports *requesting*, *ownership*, *provisioning*, *trust*, and *delegation*. Requesting identifies desires of actors. Ownership identifies the legitimate owner of a goal, a task or a resource, that has full authority on access and disposition of his possessions. Provisioning identifies actors who have the capabilities to achieve a goal, execute a task or deliver a resource. We demonstrate the use of these concepts through the design of a Medical IS (Information System) for the payment of medical care.<sup>3</sup>

**Example 1** *The Health Care Authority (HCA) is the “owner” of the goal provide medical care; that is, it is the only one that can decide who can provide it and through what process. On the other hand, Patient wants this goal fulfilled. This goal can be AND-decomposed into two subgoals: provisioning of medical care and payment for medical care. The Healthcare Provider has the capability for the provisioning of medical care, but it should wait for authorization from HCA before doing it.*

*Delegation of execution* is used to model situations where an actor (the delegator) delegates the responsibilities to achieve a goal, execute a task, or deliver a resource to another actor (the delegatee) since he does not have the capability to provide one of above by himself. It corresponds to the actual choice of the design. *Trust of execution* represents the belief of an actor (the trustor) that another actor (the trustee) has the capabilities to achieve a goal, execute a task or deliver a resource. Essentially, delegation is an action due to a decision, whereas trust is a mental state driving such decision. Tropos dependency can be defined in terms of trust and delegation [17]. Thus, a Tropos model can be seen as a particular Secure Tropos model. In order to model both functional and security requirements, Secure Tropos introduces also relations involving permission. *Delegation of permission* is used when in the domain of analysis there is a formal passage of authority (e.g. a signed piece of paper, a digital credential, etc.). Essentially, this relation is used to model scenarios where an actor authorizes another actor to achieve a goal, execute a task, or deliver a resource. It corresponds to the actual choice of the design. *Trust of permission* represents the belief of an actor that another actor will not misuse the goal, task or resource.

**Example 2** *The HCA must choose between different providers for the welfare management for executives of a public institution. Indeed, since they have a special private-law contract, they can qualify for both the INPDAP and INPDAl<sup>4</sup> welfare schemes. The INPDAP scheme requires that the Patient partially pays for medical care (with a ticket) and the main cost is directly covered by the HCA. On the contrary, the INPDAl scheme requires that the Patient pays*

<sup>3</sup> An extended description of the example is provided in [4].

<sup>4</sup> INPDAP (Istituto Nazionale di Previdenza per i Dipendenti dell’Amministrazione Pubblica) and INPDAl (Istituto Nazionale di Previdenza per i Dirigenti di Aziende Industriali) are two Italian national welfare institutes.

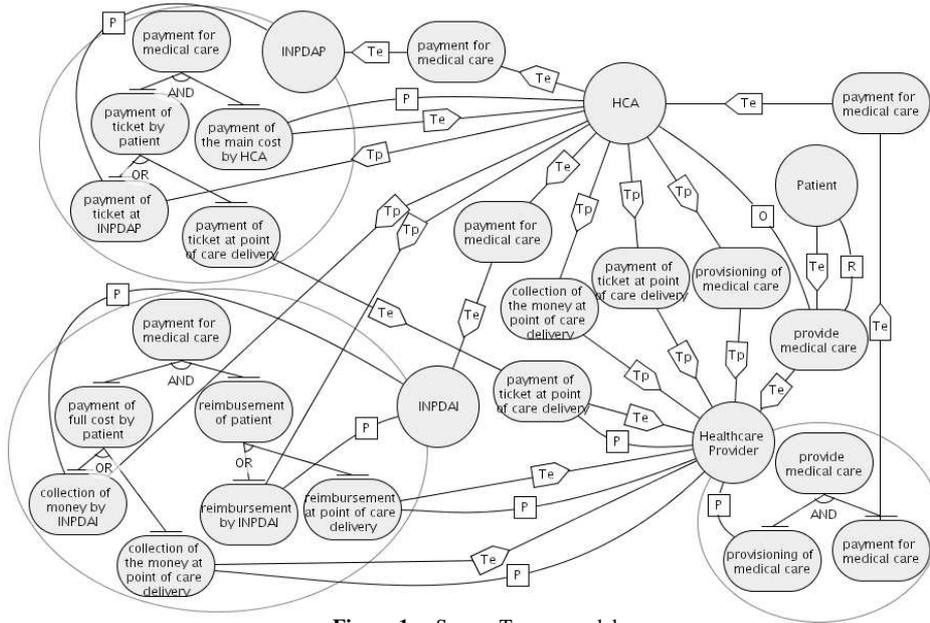


Figure 1. Secure Tropos model

in advance the full cost of medical care and then gets the reimbursement. Once an institution has decided the payment scheme, this will be part of the requirements to be passed onto the next stages of system development. Obviously, the choice of the alternative may have significant impacts on other parts of the design.

Figure 1 summarizes Examples 1 and 2 in terms of a Secure Tropos model. In this diagram, actors are represented as circles and goals as ovals. Labels **O**, **P** and **R** are used for representing ownership, provisioning and requesting relations, respectively. Finally, we represent trust of permission and trust of execution relationships as edges respectively labelled **Tp** and **Te**.

Once the modeling phase is concluded, Secure Tropos provides mechanisms for the verification of the model [16]. This means that the design process iterates over the following steps:

- model the system;
- translate the model into a set of clauses (this is done automatically);
- verify whether appropriate design or security patterns are satisfied by the model.

Through this process, we can verify the compliance of the model with desirable properties. For example, it can be checked whether the delegator trusts that the delegatee will achieve a goal, execute a task or deliver a resource (trust of execution), or will use a goal, task or resource correctly (trust of permission). Other desirable properties involve verifying whether an actor who requires a service, is confident that it will be delivered. Furthermore, an owner may wish to delegate permissions to an actor only if the latter actually does need the permission. For example, we want to avoid the possibility of having alternate paths of permission delegations. Secure Tropos provides support for identifying all these situations.

Secure Tropos has been used for modeling and analyzing real and comprehensive case studies where we have identified vulnerabilities affecting the organizational structure of a bank and its IT system [24], and verified the compliance to the Italian legislation on Privacy and Data Protection by the University of Trento [23].

### 3 Design as Planning

So far the automated reasoning capabilities of Secure Tropos are only able to check that subtle errors are not overlooked. This is rather unsatisfactory from the point of view of the designer. Whereas he may have a good understanding of possible alternatives, he may not be sure which is the most appropriate alternative for the case at hand. This is particularly true for delegations of permission that need to comply with complex privacy regulations (see [23]).

**Example 3** Figures 2(a) and 2(c) present fragments of Figure 1, that point out the potential choices of the design. The requirements engineer has identified trust relations between the HCA and INPDAP and INPDAP. However, when passing the requirements onto the next stage only one alternative has to be selected because that will be the system that is chosen. Figures 2(b) and 2(d) present the actual choices corresponding to the potential choices presented in Figures 2(a) and 2(c), respectively.

Here, we want to support the requirements engineer in the selection of the best alternative by changing the design process as follows:

- Requirements analysis phase
  - System actors along with their desires, capabilities and entitlements, and possible ways of goal decomposition are identified.
  - Trust relationships among actors both in terms of execution and permission are defined.
- Design phase
  - The space of design alternatives is automatically explored to identify delegation of execution/permission.
  - Depending on the time/importance of the goal the designer may settle for satisficing solutions [31] or ask for an optimal solution.

To support the designer in the process of selecting the best alternative we advocate a planning approach which recently has proved to be applicable in the field of automatic Web service composition [6].

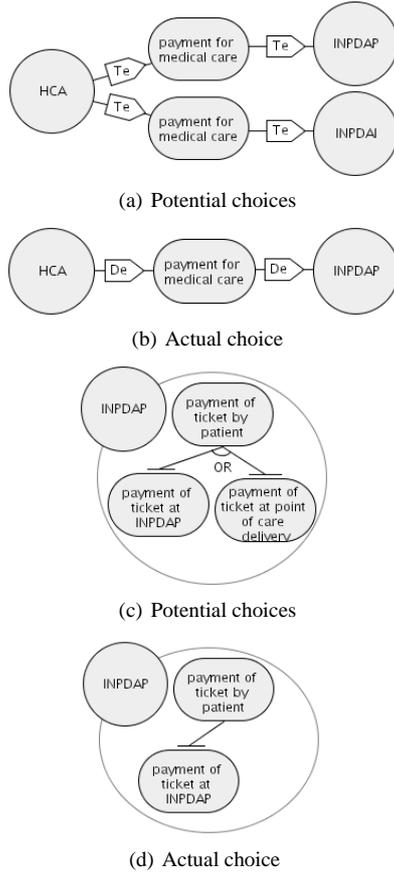


Figure 2. Design Alternatives

The basic idea behind the 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 [34, 28]. 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 consequence of the action, the system will be in a new state where the effect of the action is true. Thus, once we have described the initial state of the system, the goal that should be achieved (i.e. the desired final state of the system), and the set of possible actions that actors can perform, the solution of the planning problem is the (not necessarily optimal) sequence of actions that allows the system to reach the desired state from the initial state.

In order to cast the design process as a planning problem, we need to address the following question: *which are the “actions” in a software design?* When drawing the Secure Tropos model, the designer assigns the execution of goals from one actor to another, delegates permission and – last but not least – identifies appropriate goal refinements among selected alternatives. These are the actions to be used by the planner in order to fulfill all initial actor goals.

## 4 Planning Domain

The planning approach requires a specification language to represent the planning domain and the states of the system. Different types of logics could be applied for this purpose, e.g. first order logic is

often used to describe the planning domain with conjunctions of literals<sup>5</sup> specifying the states of the system. We find this representation particularly useful for modeling real case studies. Indeed, when considering security requirements at enterprise level, one must be able to reason both at the class level (e.g. the CEO, the CERT team member, the employee of the HR department) and at the instance level (e.g. John Doe and Mark Doe playing those roles).

The planning domain language should provide support for specifying:

- the initial state of the system,
- the goal of the planning problem,
- the actions that can be performed,
- the axioms of background theory.

Table 1 presents the predicates used to describe the *initial state of the system* in terms of actor and goal properties, and social relations among actors. We use

- **AND/OR\_decomposition** to describe the possible decomposition of a goal;
- **provides**, **requests** and **owns** to indicate that an actor has the capabilities to achieve a goal, desires the achievement of a goal, and is the legitimate owner of a goal, respectively;
- **trustexe** and **trustper** to represent trust of execution and trust of permission relations, respectively.

The desired state of the system (or *goal of the planning problem*) is described through the conjunction of predicates **done** derived from the requesting relation in the initial state. Essentially, for each **request(a,g)** we need to derive **done(g)**.

By contrast, an *action* represents an activity to accomplish a goal. We list them in Table 2 and define them in terms of preconditions and effects as follows.

**Satisfy.** The satisfaction of goals is an essential action. Following the definition of goal satisfaction given in [16], we say that an actor satisfies a goal only if the actor wants and is able to achieve the goal, and – last but not least – he is entitled to achieve it. The effect of this action is the fulfillment of the goal.

**DelegateExecution.** An actor may not have enough capabilities to achieve assigned goals by himself, and so he has to delegate their execution to other actors. We represent this passage of responsibilities through action **DelegateExecution**. It is performed only if the delegator requires the fulfillment of the goal and trusts that the delegatee will achieve it. Its effect is that the delegator does not worry any more about the fulfillment of this goal after delegating it since he has delegated its execution to a trusted actor. Furthermore, the delegatee takes the responsibility for the fulfillment of the goal and so it becomes a his own desire. Notice that we do 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.

**DelegatePermission.** In the initial state of the system, only the owner of a goal is entitled to achieve it. However, this does not mean that he wants it or has the capabilities to achieve it. On the contrary, in the system there may be some actors that want that goal and others that can achieve it. Thus, the owner could decide to authorize trusted actors to achieve the goal. The formal passage of authority takes place when the owner issues a certificate

<sup>5</sup> Let  $p$  be a predicate symbol with arity  $n$ , and  $t_1, \dots, t_n$  be its corresponding arguments.  $p(t_1, \dots, t_n)$  is called an *atom*. The expression *literal* denotes an atom or its negation.

<b>Goal Properties</b>
AND_decomposition <sub>n</sub> (g : goal, g <sub>1</sub> : goal, . . . , g <sub>n</sub> : goal)
OR_decomposition <sub>n</sub> (g : goal, g <sub>1</sub> : goal, . . . , g <sub>n</sub> : goal)
<b>Actor Properties</b>
provides(a : actor, g : goal)
requests(a : actor, g : goal)
owns(a : actor, g : goal)
<b>Actor Relations</b>
trustexe(a : actor, b : actor, g : goal)
trustper(a : actor, b : actor, g : goal)

**Table 1.** Primitive Predicates

<b>Basic Actions</b>
DelegateExecution(a : actor, b : actor, g : goal)
DelegatePermission(a : actor, b : actor, g : goal)
Satisfy(a : actor, g : goal)
AND_Refine <sub>n</sub> (a : actor, g : goal, g <sub>1</sub> : goal, . . . , g <sub>n</sub> : goal)
OR_Refine <sub>n</sub> (a : actor, g : goal, g <sub>1</sub> : goal, . . . , g <sub>n</sub> : goal)
<b>Absence of Trust</b>
Negotiate(a : actor, b : actor, g : goal)
Contract(a : actor, b : actor, g : goal)
DelegateExecution_under_suspicion(a : actor, b : actor, g : goal)
Fulfill(a : actor, g : goal)
Evaluate(a : actor, g : goal)

**Table 2.** Actions

that authorizes another actor to achieve the goal. We represent the act of issuing a permission through action `DelegatePermission` which is performed only if the delegator has the permission on the goal and trusts that the delegatee will not misuse the goal. The consequence of this action is to grant rights (on the goal) to the delegatee, that, in turn, can re-delegate them to other trusted actors.

**AND/OR\_Refine.** An important aspect of Secure Tropos is goal refinement. In particular, the framework supports two types of refinement: `OR_decomposition`, which suggests the list of alternative ways to satisfy the goal, and `AND-decomposition`, which refines a goal into subgoals which all are to be satisfied in order to satisfy the initial goal. We introduce actions `AND_Refine` and `OR_Refine`. Essentially, `AND_Refine` and `OR_Refine` represent the action of refining a goal along a possible decomposition. An actor refines a goal only if he actually need it. Thus, a precondition of `AND_Refine` and `OR_Refine` is that the actor requests the fulfillment of the initial goal. A second precondition determines the way in which the goal is refined. The effect of `AND_Refine` and `OR_Refine` is that the actor who refines the goal focuses on the fulfillment of subgoals instead of the fulfillment of the initial goal.

In addition to actions we define *axioms* in the planning domain. These are rules that hold in every state of the system and are used to complete the description of the current state. They are used to propagate actor and goal properties along goal refinement: a goal is satisfied if all its `AND`-subgoals or at least one of the `OR`-subgoals are satisfied. Moreover, axioms are used to derive and propagate entitlements. Since the owner is entitled to achieve his goals, execute his tasks and access his resources, we need to propagate actors' entitlements top-down along goal refinement.

## 5 Delegation and Contract

Many business and social studies have emphasized the key role played by trust as a necessary condition for ensuring the success of

organizations [9]. Trust is used to build collaboration between humans and organizations since it is a necessary antecedent for cooperation [1]. However, common sense suggests that fully trusted domains are simply idealizations. Actually, many domains require that actors who do not have the capabilities to fulfill their objectives, must delegate the execution of their goals to other actors even if they do not trust the delegatees. Accordingly, much work in recent years has focused on the development of frameworks capable of coping with lack of trust, sometimes by introducing an explicit notion of distrust [14, 17].

The presence (or lack) of trust relations among system actors particularly influences the strategies to achieve a goal [21]. In other words, the selection of actions to fulfill a goal changes depending on the belief of the delegator about the possible behavior of the delegatee. In particular, if the delegator trusts the delegatee, the first is confident that the latter will fulfill the goal and so he does not need to verify the actions performed by the delegatee. On the contrary, if the delegator does not trust the delegatee, the first wants some form of control on the behavior of the latter.

Different solutions have been proposed to ensure for the delegator the fulfillment of his objectives. A first batch of solutions comes from transaction cost economics and contract theories that view a contract as a basis for trust [35]. This approach assumes that a delegation must occur only in the presence of trust. This implies that the delegator and the delegatee have to reach an agreement before delegating a service. Essentially, the idea is to use a contract to define precisely what the delegatee should do and so establish trust between the delegator and the delegatee. Other theories propose models where effective performance may occur also in the absence of trust [12]. Essentially, they argue that various control mechanisms can ensure the effective fulfillment of actors's objectives.

In this paper we propose a solution for delegation of execution that borrows ideas from both approaches. The case for delegation of permission is similar. The process of delegating in the absence of trust is composed of two phases: *establishing trust* and *control*. The establishing trust phase consists of a sequence of actions, namely `Negoti-`

Planner	Release	URL
DLV <sup>κ</sup>	2005-02-23	http://www.dbai.tuwien.ac.at/proj/dlv/K/
IPP 4.1	2000-01-05	http://www.informatik.uni-freiburg.de/kochler/ipp.html
CPT 1.0	2004-11-10	http://www.cril.univ-artois.fr/vidal/cpt.en.html
SGPLAN	2004-06	http://manip.crhc.uiuc.edu/programs/SGPlan/index.html
SATPLAN	2004-10-19	http://www.cs.washington.edu/homes/kautz/satplan/
LPG-td	2004-06	http://zeus.ing.unibs.it/lpg/

**Table 3.** Comparison among planners

Requirement \ Planner	DLV <sup>κ</sup>	IPP	CPT	SGPLAN	SATPLAN	LPG-td
1		X	X	X	X	X
2				X	X	X
3	X	X	X			X

**Table 4.** Comparison among planners

ate and Contract. In **Negotiate** the parties negotiate the duties and responsibilities accepted by each party after delegation. The postcondition is an informal agreement representing the initial and informal decision of parties to enter into a partnership. During the execution of **Contract** the parties formalize the agreement established during negotiation. The postcondition of **Contract** is a trust “under suspicion” relation between the delegator and the delegatee. Once the delegator has delegated the goal and the delegatee has fulfilled the goal, the first wants to verify if the latter has really satisfied his objective. This control is performed using action **Evaluation**. Its postcondition is the “real” fulfillment of the goal. To support this solution we have introduced some additional actions (last part of Table 2) to distinguish the case in which the delegation is based on trust from the case in which the delegator does not trust the delegatee.

Sometimes establishing new trust relations might be more convenient than extending existing trust relations. A technical “side-effect” of our solution is that it is possible to control the length of trusted delegation chains. Essentially, every action has a unit cost. Therefore, refining an action into sub-actions corresponds to increasing the cost associated with the action. In particular, refining the delegation action in absence of trust guarantees that the framework first tries to delegate to trusted actors, but if the delegation chain results too long the system can decide to establish a new trust relation rather than to follow the entire trust chain.

*Need-to-know* property of a design decision states that the owner of a goal, a task or a resource wants that only the actors who need permission on its possession are authorized to access it. Essentially, only the actor that achieves a goal, executes a task or delivers a resource, and the actors that belong to the delegation of permission chain from the owner to the provider should be entitled to access this goal, task or resource. Thus, we want to obtain a plan where only the actions that contribute to reaching the desired state occur, so that if any action is removed from the plan it no longer satisfies the goal of the planning problem. This approach guarantees the absence of alternative paths of permission delegations since a plan does not contain any redundant actions.

## 6 Using the Planner

In the last years many planners have been proposed (Table 3). In order to choose one of them we have analyzed the following requirements:

1. The planner should produce solution that satisfy *need-to-know*

property by construction, that is, 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).

2. The planner should use PDDL (Planning Domain Definition Language) [15], since it is becoming the “standard” planning language and many research groups work on its implementation. In particular, the planner should use PDDL 2.2 specifications [10], since this version support features, such as derived predicates, that are essential for implementing our planning domain.
3. The planner should be available on both Linux and Windows platforms as our previous Secure Tropos reasoning tool works on both.

Table 4 presents a comparison among the planners we have considered with respect to above requirements. Based on such requirements, we have chosen LPG-td, a fully automated system for solving planning problems, supporting PDDL 2.2.

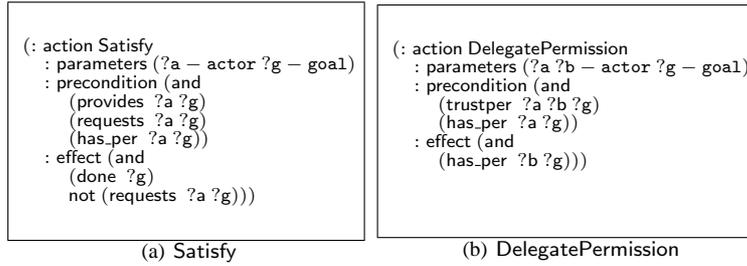
Figure 3 shows the specification of actions **Satisfy** and **DelegatePermission** in PDDL 2.2.

We have applied our approach to the Medical IS case study presented in Figure 1. The desired state of the system is obviously one where the patient gets medical care. The PDDL 2.2 specification of the planning problem is given in [4].

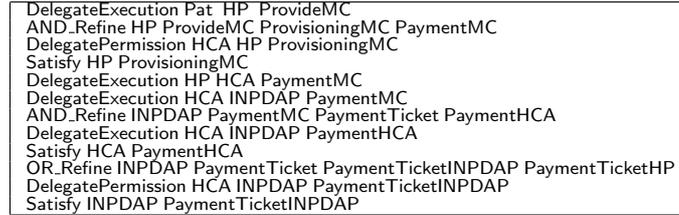
Figure 4 shows the optimal solution (i.e. the plan composed of the fewer number of actions than any other plan) proposed by LPG-td. However, this was not the first choice of the planner. Before selecting this plan, the planner proposed other two sub-optimal alternatives (see [4] for a discussion). It is interesting to see that the planner has first provided a solution with INPDAP, then a solution with INPDAI, and then, finally, a revised solution with INPDAP. A number of other experiments were conducted to test the scalability of our approach. The results are reported in [4].

## 7 Related Work

In recent years many efforts have addressed the integration of security with the system development process, in particular during early requirements analysis. In this setting, many researchers have recognized trust as an important aspect of this process since trust influences the specification of security and privacy policies. However, very few requirements engineering methodologies introduce trust concerns during the system development process. Yu et al. [36]



**Figure 3.** Actions' Specification



**Figure 4.** The optimal solution

model trust by using the concept of softgoal, i.e. a goal having no clear definition for deciding whether it is satisfied or not. However, this approach considers trust as a separate concept from security and does not provide a complete framework to consider security and trust throughout the development process. Haley et al. [18] propose to use trust assumptions, problem frames, and threat descriptions to aid requirements engineers to define and analyze security requirements, and to document the decisions made during the process.

Other approach focus on security requirements without taking into account trust aspect. van Lamsweerde et al introduce the notion of antigals for representing the goals of attackers [33]. McDermott et al. define abuse case model [26] to specify the interactions among actors, which results are harmful to some actors. Similarly, Sindre et al. define the concept of a misuse case [32], the inverse of a use case, which describes a function that the system should block.

Model Driven Architecture (MDA) approach [27], proposed by Object Management Group, is a framework for defining software design methodologies. Its central focus is on the model transformation, for instance from the platform-independent model of the system to platform-specific models used for implementation purposes. Models are usually described in UML, and the transformation is performed in accordance with the set of rules, called mapping. Transformation could be manual, or automatic, or mixed. Among the proposals on automating a software design process the one of Gamma et al. on design patterns [13] has been widely accepted. A design pattern is a solution (commonly observed from practice) to the certain problem in the certain context, so it may be thought as a problem-context-solution triple. Several design patterns can be combined to form a solution. Notice that it is still the designer who makes the key decision on what pattern to apply to the given situation.

The field of AI planning has been making advances during the last decades, and has found a number of applications (robotics, process planning, autonomous agents, Web services, etc.). There are two basic approaches to the solution of planning problems [34]. One is graph-based planning algorithms [2] in which a compact structure

called a Planning Graph is constructed and analyzed. While in the other approach [19] the planning problem is transformed into a SAT problem and a SAT solver is used. An application of the planning approach to requirements engineering is proposed by Gans et al. [14]. Essentially, they propose to map trust, confidence and distrust described in terms of *i\** models [36] to delegation patterns in a workflow model. Their approach is inspired by and implemented in ConGolog [8], a logic-based planning language. In this setting, tasks are implemented as ConGolog procedures where preconditions correspond to conditionals and interrupts. Also monitors are mapped into ConGolog procedures. They run concurrently to the other agent tasks waiting for some events such as task completion and certificate expiration. However, focus of their work is on modeling and reasoning about trust in social networks, rather than on secure design.

## 8 Conclusions

We have shown that in our extended Secure Tropos framework it is possible to automatically support the designer of secure and trusted systems also in the automatic selection of design alternatives. Our enhanced methodology allows one to:

1. Capture through a graphical notation the requirements for the organization and the system-to-be.
2. Verify the correctness and consistency of functional and security requirements by
  - completion of the model drawn by the designer with axioms (a process hidden to the designer),
  - checking the model for the satisfaction of formal properties corresponding to specific security or design patterns.
3. Automatically select alternative solutions for the fulfillment of functional and security requirements by
  - transformation of the model drawn by the designer into a planning problem (a process hidden to the designer),

- automatic identification of an alternative satisfying the goals of the various actors by means of planner.

In this paper we show that this is possible with the use of an off-the-shelf planner to generate possible designs for not trivial security requirements. Of course, we assume that the designer remains in the design loop, so the designs generated by the planner are seen as suggestions to be refined, amended and approved by the designer. In other words, the planner is a(nother) support tool intended to facilitate the design process.

Our future work includes extending the application of this idea to other phases of the design and towards progressively larger industrial case studies to see how far can we go without using specialized solvers.

## ACKNOWLEDGEMENTS

We thank Alfonso Gerevini and Alessandro Saetti for the support on the use of LPG-td. This work was partly supported by the projects RBNE0195K5 FIRB-ASTRO, RBAU01P5SS FIRB-SECURITY, 016004 IST-FP6-FET-IP-SENSORIA, 27587 IST-FP6-IP-SERENITY, 27004 IST-FP6-STREP-S3MS, 2003-S116-00018 PAT-MOSTRO, 1710SR-B/P PAT-STAMPS.

## REFERENCES

- [1] Robert Axelrod, *The Evolution of Cooperation*, Basic Books, 1984.
- [2] Avrim Blum and Merrick L. Furst, 'Fast Planning Through Planning Graph Analysis', **90**(1-2), 281–300, (1997).
- [3] Volha Bryl, Paolo Giorgini, and John Mylopoulos, 'Requirements analysis for socio-technical systems: Exploring and evaluating alternatives', Technical Report DIT-06-006, University of Trento.
- [4] Volha Bryl, Fabio Massacci, John Mylopoulos, and Nicola Zannone, 'Designing security requirements models through planning', Technical Report DIT-06-003, University of Trento.
- [5] James Caldwell, 'Moving Proofs-as-Programs into Practice', pp. 10–17, (1997).
- [6] M. Carman, L. Serafini, and P. Traverso, 'Web service composition as planning', (2003).
- [7] Jaelson Castro, Manuel Kolp, and John Mylopoulos, 'Towards Requirements-Driven Information Systems Engineering: The Tropos Project', **27**(6), 365–389, (2002).
- [8] Giuseppe de Giacomo, Yves Lespérance, and Hector J. Levesque, 'ConGolog, a concurrent programming language based on the situation calculus', **121**(1-2), 109–169, (2000).
- [9] Peter Drucker, *Managing the Non-Profit Organization: Principles and Practices*, HapperCollins Publishers, 1990.
- [10] Stefan Edelkamp and Jorg Hoffmann, 'Pddl2.2: The language for the classical part of the 4th international planning competition', Technical Report 195, University of Freiburg, (2004).
- [11] Thomas Ellman, 'Specification and Synthesis of Hybrid Automata for Physics-Based Animation', pp. 80–93, (2003).
- [12] Michael J. Gallivan, 'Striking a balance between trust and control in a virtual organization: a content analysis of open source software case studies', **11**(2), 277, (2001).
- [13] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995.
- [14] G. Gans, M. Jarke, S. Kethers, and G. Lakemeyer, 'Modeling the Impact of Trust and Distrust in Agent Networks', pp. 45–58, (2001).
- [15] M. Ghallab, A. Howe, C. Knoblock, D. McDermott, A. Ram, M. Veloso, D. Weld, and D. Wilkins, 'PDDL – The Planning Domain Definition Language', (1998).
- [16] Paolo Giorgini, Fabio Massacci, John Mylopoulos, and Nicola Zannone, 'Modeling Security Requirements Through Ownership, Permission and Delegation', pp. 167–176, (2005).
- [17] Paolo Giorgini, Fabio Massacci, John Mylopoulos, and Nicola Zannone, 'Modelling Social and Individual Trust in Requirements Engineering Methodologies', volume 3477, pp. 161–176, (2005).
- [18] Charles B. Haley, Robin C. Laney, Jonathan D. Moffett, and Bashar Nuseibeh, 'Using Trust Assumptions with Security Requirements', **11**, 138–151, (2006).
- [19] Henry Kautz and Bart Selman, 'Planning as satisfiability', pp. 359–363. John Wiley & Sons, Inc., (1992).
- [20] Emmanuel Letier and Axel van Lamsweerde, 'Reasoning about partial goal satisfaction for requirements and design engineering', **29**(6), 53–62, (2004).
- [21] Niklas Luhmann, *Trust and Power*, Wisley, 1979.
- [22] Zohar Manna and Richard Waldinger, 'A Deductive Approach to Program Synthesis', *ACM Transactions on Programming Languages and Systems*, **2**(1), 90–121, (1980).
- [23] Fabio Massacci, Marco Prest, and Nicola Zannone, 'Using a Security Requirements Engineering Methodology in Practice: The compliance with the Italian Data Protection Legislation', **27**(5), 445–455, (2005).
- [24] Fabio Massacci and Nicola Zannone, 'Detecting Conflicts between Functional and Security Requirements with Secure Tropos: John Rusnak and the Allied Irish Bank', Technical Report DIT-06-002, University of Trento, (2006).
- [25] Mihhail Matskin and Enn Tyugu, 'Strategies of Structural Synthesis of Programs and Its Extensions', **20**, 1–25, (2001).
- [26] J. McDermott and C. Fox, 'Using Abuse Case Models for Security Requirements Analysis', pp. 55–66, (1999).
- [27] Object Management Group. Model Driven Architecture (MDA). <http://www.omg.org/docs/ormsc/01-07-01.pdf>, July 2001.
- [28] Joachim Peer, 'Web Service Composition as AI Planning - a Survey', Technical report, University of St. Gallen, (2005).
- [29] Steve Roach and Jeffrey Baalen, 'Automated Procedure Construction for Deductive Synthesis', **12**(4), 393–414, (2005).
- [30] Ragnhild Kobro Runde and Ketil Stølen, 'What is model driven architecture?', Technical Report UIO-IFI-RR304, Department of Informatics, University of Oslo, (March 2003).
- [31] Herbert A. Simon, *The Science of the Artificial*, MIT Press, 1969.
- [32] Guttorm Sindre and Andreas L. Opdahl, 'Eliciting security requirements with misuse cases.', **10**(1), 34–44, (2005).
- [33] Axel van Lamsweerde, Simon Brohez, Renaud De Landtsheer, and David Janssens, 'From System Goals to Intruder Anti-Goals: Attack Generation and Resolution for Security Requirements Engineering', pp. 49–56, (2003).
- [34] Daniel S. Weld, 'Recent Advances in AI Planning', *AI Magazine*, **20**(2), 93–123, (1999).
- [35] Rosalinde Klein Woolthuis, Bas Hillebrand, and Bart Nooteboom, 'Trust, Contract and Relationship Development', *Organization Studies*, **26**(6), 813–840, (2005).
- [36] Eric S. K. Yu and Lin Liu, 'Modelling Trust for System Design Using the i\* Strategic Actors Framework', volume 2246, pp. 175–194, (2001).