Autom Softw Eng (2007) 14: 341–364
DOI 10.1007/s10515-007-0013-5

Computer-aided Support for Secure Tropos

Fabio Massacci · John Mylopoulos · Nicola Zannone

Received: 7 August 2006 / Accepted: 27 June 2007 / Published online: 1 August 2007

Abstract In earlier work, we have introduced Secure Tropos, a requirements engineering methodology that extends the Tropos methodology and is intended for the design and analysis of security requirements. This paper briefly recaps the concepts proposed for capturing security aspects, and presents an implemented graphical CASE tool that supports the Secure Tropos methodology. Specifically, the tool supports the creation of Secure Tropos models, their translation to formal specifications, as well as the analysis of these specifications to ensure that they comply with specific security properties. Apart from presenting the tool, the paper also presents a two-tier evaluation consisting of two case studies and an experimental evaluation of the tool's scalability.

Keywords Security requirements engineering · CASE tools · Automated reasoning

1 Introduction

Requirement Engineering is the phase of the software development process that aims at understanding the organizational context of a system, the goals of organizational and system actors, and social relationships among them (Nuseibeh and Easterbrook 2000). This phase is particularly critical, because misunderstandings may lead to expensive errors during later phases.

F. Massacci · J. Mylopoulos · N. Zannone (✉)
Department of Information and Communication Technology, University of Trento, Trento, Italy
e-mail: zannone@dit.unitn.it

F. Massacci
e-mail: massacci@dit.unitn.it

J. Mylopoulos
e-mail: jm@dit.unitn.it

Graphical modeling has been recognized as an important aspect in Software Engineering in general, and Requirements Engineering in particular, because it facilitates and promotes understanding between designers and stakeholders. Graphical models are thus being widely used to design systems and capture their functional properties, as well as those of the environment within which the system-to-be will operate.

Unfortunately, graphical notations are often informal, and generally lack the expressiveness of logic-based requirements modeling languages. This issue has spurred the development of many formal frameworks that combine elements of graphical notations and logic. In addition, there have been many proposals for verifying requirements, ranging from automated reasoning (Fickas and Nagarajan 1988; Maiden and Sutcliffe 1992) to requirements animation (Gravell and Henderson 1996). Together with analysis techniques, CASE tools have been developed in order to support such techniques and drive requirements elicitation and analysis.

Recent years have seen an increasing awareness that security plays a key role within organizations and their IT systems. Consequently, many research efforts are focusing on the introduction of security concerns into the system development process (Basin et al. 2006; Jürjens 2004), also, modeling and analysis of security requirements along-side functional requirements (Haley et al. 2005; Liu et al. 2003; McDermott and Fox 1999; Sindre and Opdahl 2005; van Lamsweerde 2004).

These proposals tackle the problem of designing secure systems from different perspectives. Some proposals model attackers along with their capabilities (Liu et al. 2003; van Lamsweerde 2004), whereas others define the undesirable behaviors that a system should prevent (McDermott and Fox 1999; Sindre and Opdahl 2005). There are also proposals that focus on security related features such as confidentiality and access control (Basin et al. 2006; De Landtsheer and van Lamsweerde 2005; Jürjens 2004; Onabajo and Jahnke 2006). However, most of such proposals focus on system security aspects, rather than social and organizational ones.

In the context of socio-technical systems—reflected in our past work—security requirements are mostly social requirements rather than technical solutions (Giorgini et al. 2006). To understand the problem of security engineering we need to model and analyze organizational settings, in terms of relationships between relevant actors, including the system-to-be. Modeling only digital protection mechanisms is not sufficient. Indeed, several studies have revealed how security is often compromised by exploiting weaknesses at the interface between procedures and policies adopted by an organization and the system that support them (Anderson 1994; House of Lords 1999; Promontory Financial Group 2003).

The Tropos methodology (Bresciani et al. 2004) is an agent-oriented software engineering methodology intended to support the modeling and analysis of both the system-to-be and its organizational environment through different phases of the system development process. One of its main features is the prominent role given to early requirements analysis phase that concerns the understanding of the domain by studying the organizational context within which the system-to-be will eventually operate. The main advantage in having such a phase is that one can capture not only the "what" or the "how", but also the "why" system functionalities are required. This methodology has already been used to model security requirements by offering facilities for analyzing threats, vulnerabilities, and countermeasures (Liu et al. 2003).

This approach supports the representation of design decisions relevant to security by modeling internal and external attackers along with their goals and design solutions that prevent their fulfillment. However, Tropos lacks fundamental concepts necessary to deal with certain aspects of security (Giorgini et al. 2006).

In (Giorgini et al. 2005b), the authors propose Secure Tropos, an agent-oriented security requirements engineering methodology, that extends Tropos with concepts specific to security and supports formal analysis. The methodology aims at assisting system designers in the acquisition of requirements and the verification of their compliance with security properties. Though it is possible within this framework to model and analyze the behavior and objectives of attackers similarly to the work in (Liu et al. 2003), the main focus of formal analysis in this paper is the exploitation of security-specific concepts to verify the correctness and consistency of procedures and policies in their use of assets. In particular, it provides support for verifying that actor objectives will be fulfilled and that their entitlements will not be misused. Moreover, the framework provides facilities for verifying the compliance of organizational policies with the need-to-know principle that is a fundamental requirement established by privacy legislation in many countries (e.g., Data Protection Directive, 95/46/EC). Together with a modeling language and a methodological framework, the authors propose a formal framework based on Answer Set Programming (ASP) (Leone et al. 2006) to assist designers during security requirements verification.

Though several existing Requirements Engineering methodologies are often coupled with CASE tools, such coupling is far less frequent for Security Requirements Engineering. The application of the Secure Tropos methodology to some real case studies (Asnar et al. 2006; Massacci and Zannone 2006; Massacci et al. 2005) has shown the need for such tools. This application has also revealed the need for such tools to not only support requirements engineers through requirements elicitation, but also to offer formal methods for requirements analysis.

This paper presents ST-Tool, a CASE tool developed to support the Secure Tropos methodology. The main objectives behind the design of the tool are:

- *Graphical environment*: a framework to assist requirements engineers in the creation of graphical models of early security requirements using Secure Tropos concepts.
- *Formalization*: support for translating graphical models into formal specifications.
- *Analysis facilities*: a front-end to state-of-the-art, off-the-shelf ASP solvers that support the analysis of these specifications to ensure that desirable security properties are satisfied.

In the next section (Sect. 2) we provide an overview of Secure Tropos along with the concepts it is founded on. We then present the tool supporting this methodology (Sect. 3) and describe how the tool assists designers during requirements modeling and analysis (Sect. 4). In order to evaluate the tool, we next present the application of the methodology and tool to industrial case studies, as well as experimental results on its scalability (Sect. 5). Finally, we discuss related work (Sect. 6) and conclude the paper with discussion on future directions for this research (Sect. 7).

For pragmatic reasons, the scope of the evaluation of the tool does not include usability, nor we have tried to evaluate the effectiveness of the tool in supporting

requirements engineers in the design of secure systems. Moreover, although the paper clearly identifies what security properties can be formally verified by the ST-Tool, it does not include a general discussion of security properties that are not handled by our tool.

2 Secure Tropos

Tropos (Bresciani et al. 2004) is an agent-oriented software engineering methodology tailored to model both the system-to-be and its organizational environment. Secure Tropos (Giorgini et al. 2005b) extends Tropos in order to model and analyze security requirements alongside functional requirements. The methodology provides a requirements analysis process that drives system designers from the acquisition of requirements up to their verification.

Secure Tropos adopts from Tropos and i* (Yu 1995) the concepts of actor, goal, task, resource, and introduces the concepts of objective, entitlement, capability, delegation and trust. An *actor* is an intentional entity that performs actions to achieve goals. A *goal* represents a strategic interest of an actor. A *task* represents a course of actions for satisfying a goal. A *resource* represents a physical or an informational entity. For the sake of brevity, we use the term "service" to refer to a goal, task, or resource when it is not necessary to distinguish them. Objectives, entitlements and capabilities of actors are modeled through relationships between an actor and a service, namely request, own, and provide. *Request* identifies goals intended to be achieved, tasks intended to be executed or resources required by actors; *own* represents the authority of controlling the achievement of a goal, execution of a task, or delivery of a resource; and *provide* represents the capability of achieving a goal, executing a task, or furnishing a resource.

Moreover, Secure Tropos uses the notion of delegation of permission and delegation of execution. *Delegation of permission* indicates that one actor authorizes another actor to achieve a goal, execute a task, or furnish a resource. As consequence, the delegate is entitled to achieve the goal, execute the task, or furnish the resource. Moreover, the delegate may also re-delegate the granted permission (or part of it) to other actors. *Delegation of execution* (or *execution dependency*) indicates that one actor appoints another actor to achieve a goal, execute a task, or furnish a resource. As consequence, the delegate is responsible to achieve the goal, execute the task, or furnish the resource. Moreover, the delegate may also re-delegate the assigned responsibility (or part of it) to other actors. System designers might need to model situations where an actor must delegate the execution of his objectives or the permission on his entitlements to actors he does not trust. Thus, it is necessary to separate the concept of trust from the concept of delegation. In particular, the concepts of *trust of permission* is used to model the expectation of an actor about the fair behavior of another actor and *trust of execution* to model the expectation of an actor about the achievement of a goal, execution of a task, or delivery of a resource by another actor.

The above constructs allow designers to capture the requirements model of organizations together with their IT systems. In the graphical representation of this model, objectives, entitlements, and capabilities are represented using request (**R**),

ownership (**O**), and provide (**P**) relations, respectively. Permission delegations are represented with edges labeled **Dp** and execution dependency with edges labeled **De**. Finally, trust of permission and trust of execution are represented with edges labeled **Tp** and **Te** respectively.

The Secure Tropos methodology proposes the following modeling activities for representing the requirements model:

Actor modeling consists of identifying and analyzing domain stakeholders and system actors along with their objectives, entitlements and capabilities.

Trust modeling consists of modeling the expectation of actors about the performance and fair behavior of other actors on a certain goal, task, or resource. Such expectations are modeled using trust of execution and trust of permission links.

Execution Dependency modeling consists of identifying transfers of responsibilities from an actor to another. Such transfers are modeled using delegation of execution links.

Permission Delegation modeling consists of identifying transfers of authority from an actor to another. Such transfers are modeled using delegation of permission links.

Their outcome is a number of graphical models that together constitute a requirements model. In particular, each diagram type corresponds to a different *view* of the requirements model, respectively an *actor*, *trust*, *execution dependency*, and *permission delegation view*.

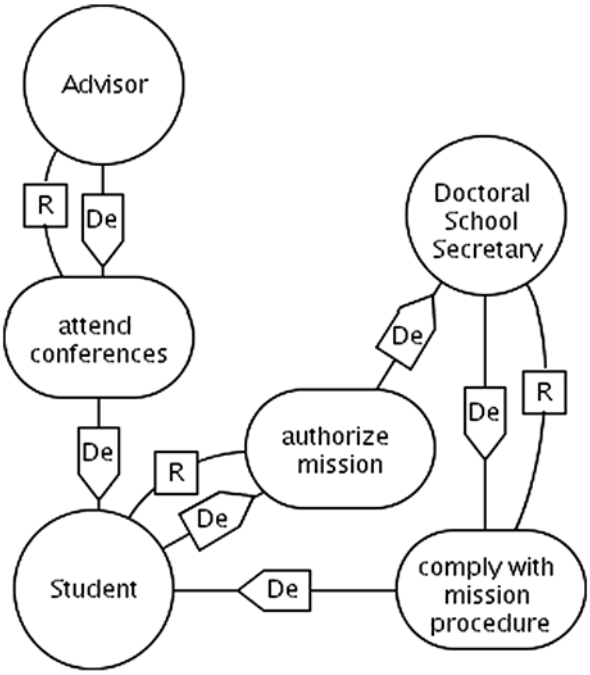
Example 1 An adviser wants to have his PhD students attend conferences in order to present their research work, and delegates the execution of this goal to them. PhD students need authorization for the mission and request it from the Doctoral School Secretary. The Doctoral School Secretary is appointed by the University to comply with University procedures and regulations. These duties include checking that every student is in compliance with University and Doctoral School regulations and procedures. Thereby, the Doctoral School Secretary requires students to ensure compliance with mission procedures. The requirements model is presented in Fig. 1.

Goal/Task modeling refines and enriches a requirements model with further details. This modeling activity is conducted from the perspective of single actors using AND/OR refinement, means-end analysis, and contribution analysis (Bresciani et al. 2004).

A graphical representation of goal/task modeling is given through *goal diagrams*.

Example 2 Figure 2 refines the model presented in Fig. 1 by focusing on the mission procedure for PhD students according to ICT International Doctoral School's regulation at the University of Trento (UNITN) – see "Regulations & Procedures" at http://ict.unitn.it/ict/services/faq.xml. In particular, to achieve the goal comply with mission procedure, the student has to get informal approval for the mission for which he depends on his adviser, fill request mission, and (possibly) get advance payment

Fig. 1 Requirements model



and get payment of expenses for which he depends on the Administration Department. In order to achieve goal fill request mission, the student has to provide mission information, choose means of transportation, and provide funds information. The achievement of this goal requires the fulfillment of goals decide funding and require authorization for the use of funding. Depending on the chosen funding account, the student shall ask for authorization either from the Doctoral School Dean or his adviser. To achieve their duties, the Doctoral School Secretary and Administration Department need some documents from the student, such as mission request form, advance reimbursement form, and payment of expenses form with original receipts.

Once designers have elaborated a first version of a requirements model, the methodology supports them in verifying its compliance with specific security properties. In particular, the framework allows one to verify the availability, authorization, and privacy of the designed system using the properties presented in Table 1. These properties partially cover the security and privacy requirements identified in a number of authoritative classifications from widely accepted ISO standards, such as ISO 17799 and ISO 15408, a recent survey by Avizienis et al. (2004), the classical paper by Saltzer and Schroeder (1975), and a seminal paper on Hippocratic databases (Agrawal et al. 2002). However, a detailed discussion about the positioning of the properties we can verify with respect to these classifications go beyond the scope of this paper.

Security properties may be classified as safety properties and liveness properties. Intuitively, *safety properties* are properties stating "nothing bad will happen" and *liveness properties* are properties stating "something good will happen" (Alpern

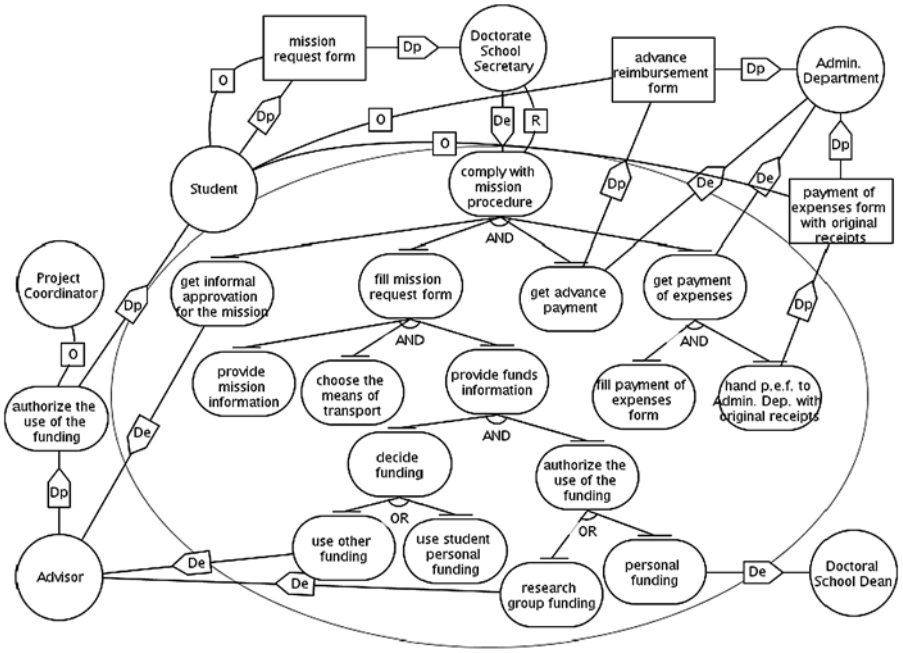


Fig. 2 The mission procedure at UNITN

and Schneider 1986). Most security properties rely on the fact that attackers cannot compromise the system. Therefore, they can be characterize as safety properties. For instance, authentication, access control, confidentiality, integrity and non-repudiation are safety properties (Germeau and Leduc 1997). Each of these security services require that a particular situation will not occur. The only security liveness property is availability and, in particular, non-denial of service (Bauer et al. 2002; Germeau and Leduc 1997). Nonetheless, this classification is not rigid. Indeed, it is known that any property, including security properties, can be expressed as the conjunction of a safety and a liveness property (Bauer et al. 2002; Schneider 1987). The properties in Table 1 can roughly be classified as follows: Pro1, Pro4, Pro5, Pro6 and Pro10 are safety properties and Pro2, Pro3, Pro7, Pro8, and Pro9 are combinations of safety and liveness properties. For instance, Pro5 is a safety property because there is a bad thing that may occur (e.g., an untrusted actor can have permission to achieve a goal).

The development of systems with high security needs requires methodologies that support the definition and verification of defense-in-depth strategies, which are practical strategies for achieving information assurance (National Security Agency 2002, Chap. 4). Design for defense-in-depth can be supported by this or other methodologies by having the analyst explicitly add or remove trust assumptions (trust relations in our case) and verify that he still gets a successful verification of the security properties. In our case this simply means that the designer needs to remove a trust relation or a combination of these relations and check that all properties (e.g., confidence in satisfaction) still hold, then restore those relations, remove others and check the

Table 1 Desirable properties

Availability	
Pro1	Actors delegate the execution of their (direct or indirect) objectives only to actors that they trust.
Pro2	Requesters can satisfy their objectives; that is, they have assigned their objectives to actors that have the capabilities to achieve them.
Pro3	Requesters are confident their objectives will be satisfied; that is, they have assigned their objectives to actors that have the capabilities to achieve them and are trusted.
Authorization	
Pro4	Actors delegate permissions on their (direct or indirect) entitlements only to actors they trust.
Pro5	Owners are confident that their entitlements are not misused; that is, permission on their entitlements is assigned only to actors they trust.
Pro6	Actors, who delegate permissions to achieve a goal, execute a task, or furnish a resource, have the right to do so.
Availability & Authorization	
Pro7	Requesters can achieve their objectives; that is, they have assigned their objectives to actors that have both the permissions and capabilities to achieve them.
Pro8	Requesters are confident to achieve their objectives; that is, they have assigned their objectives to actors that have both the permissions and capabilities to achieve them and are trusted.
Pro9	Providers have the permissions necessary to accomplish assigned duties.
Privacy	
Pro10	Permission has been granted to actors who actually need it to perform their duties. This is an important property to fulfill EU legislative privacy requirements on need-to-know.

properties again. Since the decision on whether any actor is trustworthy is up to the designer, such a procedure will likely remain manual though it is possible to add rules that define single points of failure in a trust chain.

3 ST-tool

The Secure Tropos methodology is supported by the ST-Tool (http://sesa.dit.unitn.it/sttool/). The tool consists of three parts: the modeling kernel, the visual model interface, and the reasoning interface. In Fig. 3, the modules of ST-Tool are shown, along with their interrelations.

3.1 Visual modeling

To manage visual editing features and data management consistency at the same time, we have adopted an architecture that includes a graphical layer and a data layer. At the graphical layer, models are presented as graphs where actors and services are nodes, and relations are arcs. Each visual object refers to a data object. The collection of data

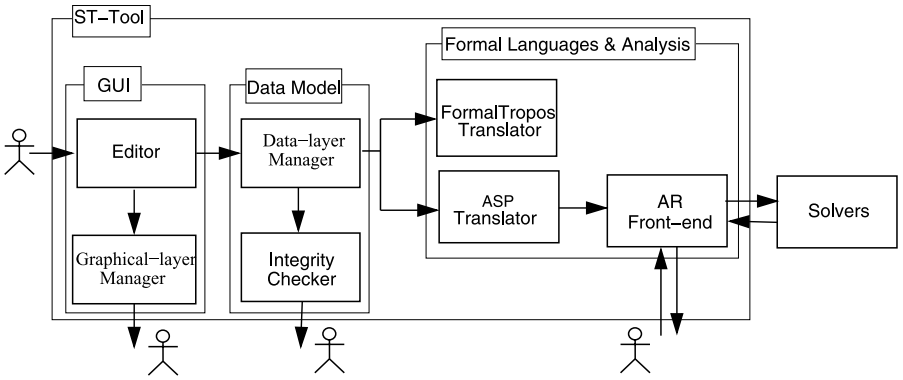


Fig. 3 Architecture overview

objects is managed by the data layer. Distinguishing the data layer from the graphical layer allows designers, for example, to draw Tropos models and then build on top of them Secure Tropos models by reusing common concepts. Moreover, it permits the association of more than one graphical object to the same data object. This feature is essential for dealing with situations where a service is re-delegated by some actor. It should be noted that the Tropos graphical notation is similar, but not identical, to a subset of the Secure Tropos notation.

Implementation-wise, the ST-Tool provides a graphical user interface (GUI), through which all its graphical components are managed. The screen is divided into four main areas: the menu of entities and relations at the top, the graphical editor in the middle, the menu for choosing different representations of requirements models at the bottom, and the property viewer and editor at the left (Fig. 4). The GUI's key component is the Editor module. This module allows designers to visually insert, edit or remove graphical objects in the graphical layer by selecting the desired entity or relation from the top menu, and specifying entity and relation attributes in the data layer on the left menu. In particular, for every Tropos element, it is possible to specify temporal properties according to the syntax of Formal Tropos (Fuxman et al. 2004) by selecting the corresponding panel on the left menu.

A second component of the tool is the Graphical-layer Manager (GM) module that manages graphical objects. For instance, it supports goal modeling by associating a goal diagram with each actor. Correlated with this feature, GM supports two types of collapsing nodes, namely collapsing services and actors. When a service is collapsed, its sub-services, decomposition arcs and relations having subservices as dependum are hidden. When an actor is collapsed, the entire rationale of that actor is hidden. GM also allows designers to activate one or more views of the requirements model (i.e., dependency diagram for Tropos (Bresciani et al. 2004), and trust, execution dependency, and permission delegation views for Secure Tropos) at the same time. Essentially, when a view is activated/deactivated, all elements related to that particular view are shown/hidden. For instance, deactivating trust view will hide all trust relation links in the requirements model.

The Data-layer Manager (DM) module is responsible for building and maintaining data corresponding to graphical objects. For example, DM manages misalignment

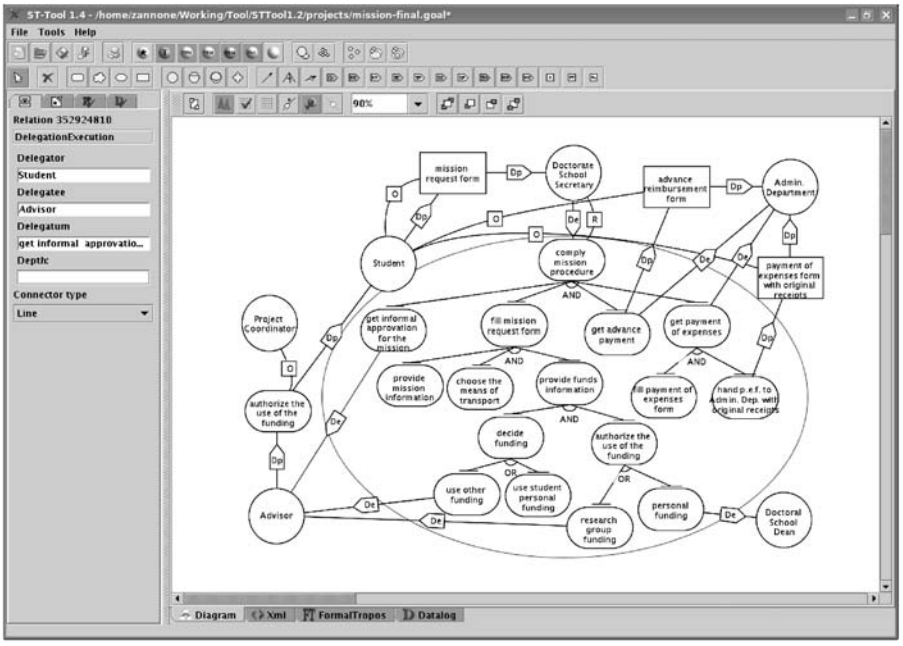


Fig. 4 ST-tool graphical user interface

between social relations and their graphical representation. Essentially, this module rebuilds Tropos and Secure Tropos relations by linking the appropriate graphical objects (e.g., two trust links, two actor nodes representing the trustor and the trustee, and a service node representing the trustum) to the same data object (the trust relation).

Finally, the tool uses the Integrity Checker module to detect errors during the modeling phase. In particular, this module is responsible to identify designers' misapplications such as "isolated nodes" (i.e. services not involved in any relation), "orphan relations" (i.e. relations where an arc is missing), and bad-typed relations, by parsing the requirements model stored in the DM module.

3.2 Automated reasoning support

In order to perform formal analysis, the ST-Tool supports the automatic transformation of graphical models into formal specifications. Currently, two logics are supported: temporal logic for behavioral specification (Fuxman et al. 2004) and Answer Set Programming (ASP) for security verification (Giorgini et al. 2005b). These transformations are automatically performed, respectively, by two modules: the Formal Tropos Translator module and the ASP Translator module. These modules are responsible for the translation of graphical models into the corresponding formal specifications. The resulting specifications can be displayed by selecting the corresponding panel in the bottom menu (Fig. 5).

Supporting languages based on first order logic and languages based on temporal logic allows system designers to perform different types of analysis, so they can

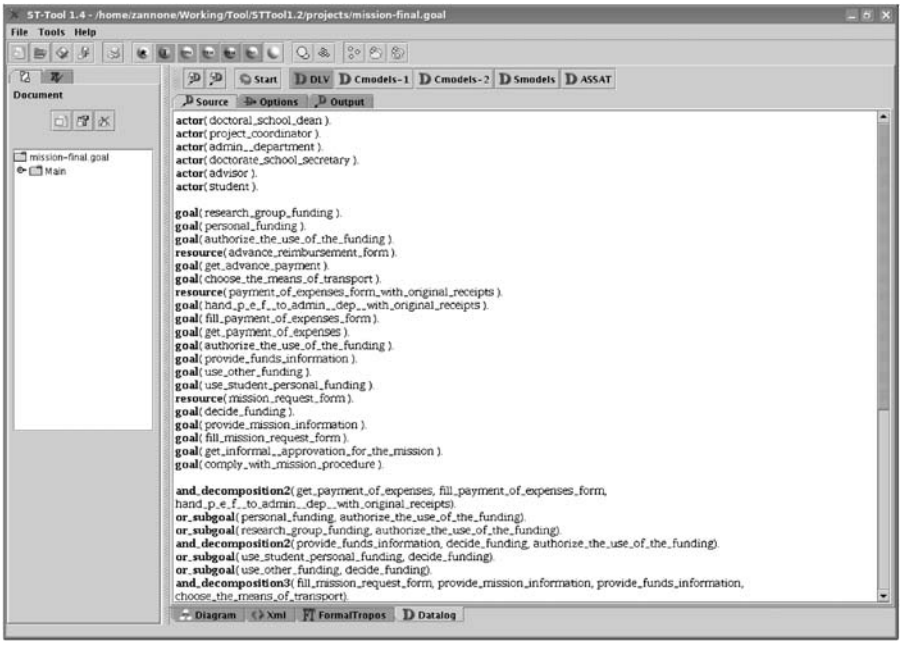


Fig. 5 ST-tool: the ASP specification

evaluate the system-to-be along with the environment where it will act from different perspectives. As already mentioned, in this paper we focus on the formal analysis performed using the ASP paradigm. Basically, this analysis is addressed to verify the compliance of system requirements with the properties given in Table 1.

Apart from choosing different formal languages, there are also different types of analysis one may want to perform. For example, one may focus on verifying the correctness of the current setting of an organization. For the analysis of an enterprise-wide privacy policy, we cannot simply reason at the level of generic concepts (e.g., "the user", "the General Director", "the member of the CERT team"), but we must instantiate these concepts to specific individuals playing those roles (Massacci et al. 2005). Towards this end, ST-Tool supports the instantiation of the requirements model at the organizational level with an arbitrary number of instances by associating a number of individuals with a role.

The Automated Reasoning Front-end (ARF) module provides designers with functionality tailored to complete and check requirements models expressed in form of ASP specifications using different external ASP solvers. This permits designers to select the properties to be verified and the information to be visualized in the output (Fig. 6). This information allows designers to understand the "where" and "why" of system vulnerabilities. Designers may also need to specify and derive additional information about the domain under analysis. To this end, the ARF module provides support to add rules, properties, or facts that will be analyzed together with the intuitive description of the system, axioms, and selected properties. The tool supports the use of different ASP solvers, namely ASSAT (Lin and Zhao 2002), Cmodels (Lierler

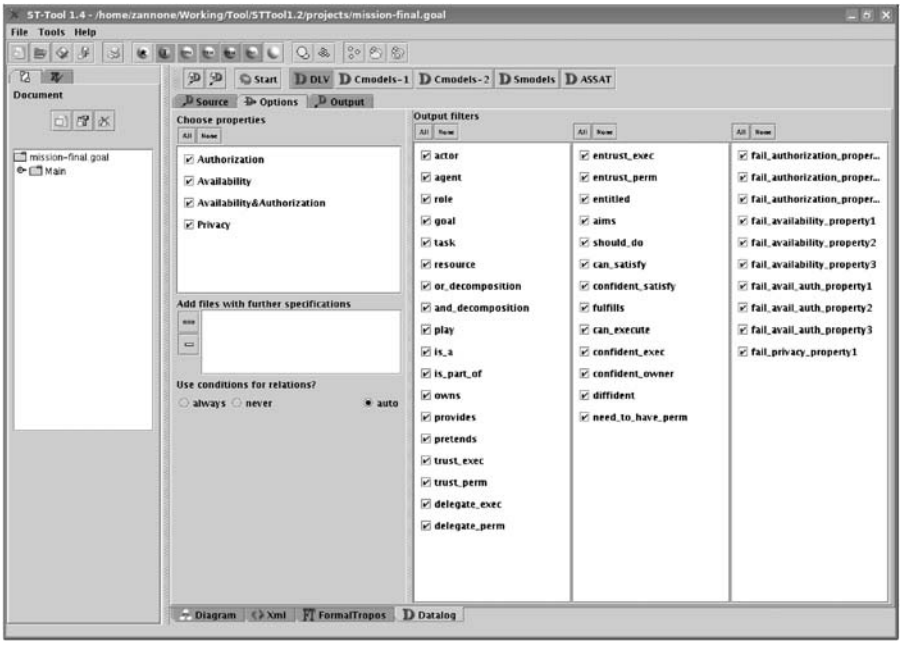


Fig. 6 Automated reasoning front-end

2005), Smodels (Niemelä et al. 2000), and DLV (Leone et al. 2006). Cmodels and ASSAT use SAT solvers as inference engines, while Smodels uses a general-purpose answer set solver. All these solvers work with grounded logic programs generated by Lparse (Niemelä and Simons 1996). Lparse grounds a logic programs by transforming it into an equivalent ground logic program (i.e., a logic program whose rules do not containing variables) where equivalence is defined as having the same set of stable models. Finally, DLV is a deductive database system. The ARF module is responsible for passing the ASP specification corresponding to the graphical requirements model to the chosen ASP solver for verifying consistency. Once the solver completes its task, it returns the minimal Herbrand model (i.e., a set of ground literals) satisfying the program itself (Gelfond and Lifschitz 1991). This model is parsed and presented in a user-readable format by the ARF module.

In order to optimize the analysis, we have implemented the formal framework underlying Secure Tropos as a locally stratified logic program (Gelfond and Lifschitz 1988). Such programs consist of rules that do not have arbitrary recursion through negation. Specifically, predicates of locally stratified programs can be partitioned into disjoint sets called *strata* such that there is no negative dependency between predicates in the same stratum. A predicate p depends on predicate q if q occurs in the body of the rule in which p occurs as head. A locally stratified program has to satisfy two conditions:

1. if predicate p is at stratum i and depends positively on predicate q, then q must be in a stratum j such that $j \leq i$, and

2. if predicate p is at stratum i and depends negatively on predicate q, then q must be in a stratum j such that $j < i$.

Intuitively, intensional predicates are completely computed without relying via negation on other predicates that are not in a lower stratum. This restriction implies that recursive iterations can be resolved in a finite number of steps. Locally stratified logic programs have a unique stable model and a well-founded semantics (Gelfond and Lifschitz 1988). Moreover, van Gelder (1989) has proved that the unique stable model can be computed in quadratic time complexity. While the use of locally stratified logic programs may seem like a limitation in expressiveness, they were sufficient for expressing the semantics of Secure Tropos concepts.

4 Requirements analysis support

This section presents how the ST-Tool assists system designers in the use of the Secure Tropos methodology. The main activities supported by the tool are: (1) requirements elicitation and modeling; (2) model transformation; (3) formal requirements analysis.

4.1 Requirements elicitation and modeling

Requirements elicitation and modeling activities aim to capture requirements and represent them in terms of Secure Tropos diagrams.

During requirements elicitation, domain stakeholders and system actors are identified along their properties; their inter-dependencies are also captured and represented in terms of social relations. Requirements specifications are usually provided by stakeholders in natural language. We have defined a requirements collection schema designed to capture requirements in a semi-structured way (Asnar et al. 2006). This form-based document is intended to bridge the gap between requirements specified in natural language and the formal requirements specification. It provides a number of tables that drive requirements engineers during requirements elicitation.

Once the requirements collection schema has been completed, requirements modeling proceeds using the graphical Editor. Indeed, there is a one-to-one relation between Secure Tropos concepts and the tables in the template.

4.2 Model transformation

The semantics of all Secure Tropos concepts are defined using the Answer Set Programming (ASP) paradigm (Gelfond and Lifschitz 1991). Roughly speaking, the ASP paradigm is a variant of Datalog with negation as failure. This paradigm supports specifications expressed in terms of facts and Horn clauses, which are evaluated using the stable model semantics. A fact consists of a relation symbol, called *predicate*, together with an appropriate number of well-formed arguments. In ASP, as in Datalog, predicates are distinguished into two types: extensional and intensional. A predicate is extensional if it does not appear on the left-hand side of any clause. Extensional

(1) owns(*student*, *mrf*)
(2) owns(*student*, *arf*)
(3) owns(*student*, *pef*)
(4) owns(*project_coordinator*, *authorize_the_use_of_the_funding*)
(5) delegate_perm(*student*, *dss*, *mrf*)
(6) delegate_perm(*student*, *ad*, *arf*)
(7) delegate_perm(*student*, *ad*, *pef*)
(8) delegate_perm(*advisor*, *student*, *authorize_the_use_of_the_funding*)

Fig. 7 Extensional description of the system in ASP

predicates represent primitive concepts in Secure Tropos. All other predicates are called intentional, and they are used for requirements verification.

The ST-Tool supports the translation of Secure Tropos diagrams to ASP clauses through the ASP module.

Example 3 Figure 7 shows the list of facts (limited to permission) corresponding to the intuitive description of the system given in Example 2. We use literal owns(a, s) to represent that actor a is the legitimate owner of service s, and delegate_perm(a, b, s) for modeling the transfer of authority on service s from actor a to actor b.

4.3 Formal requirements analysis

To verify the correctness of requirements, the framework supports formal analysis of rules and constraints (i.e., rules where the head is empty). Rules (or *axioms*) are used to complete specifications in order to derive the information needed for requirements verification. For instance, designers need to identify who is entitled to achieve goals, perform tasks and access resources. To this intent, Secure Tropos uses the following axioms (Giorgini et al. 2005b):

(Ax1) has_perm(A,S) ← owns(A,S)
(Ax2) has_perm(B,S) ← delegate_ perm(A,B,S) ∧ has_perm(A,S)

Ax1 states that actors are entitled to access their own services, and Ax2 that actors authorized by someone who is entitled to access the service, are in turn entitled to access that service. It should be noted that in this paper, we have reported a simplified version of axioms where delegation is assumed to be transitive. We refer to (Giorgini et al. 2006) for a discussion on how delegation depth and conditions can be used to control (re-)delegation.

Example 4 Applying such axioms to our example, one can infer that the student is entitled to access all her own forms, the Doctorate School Secretary is entitled to access student mission request form, and the Administrative Department is entitled to access student advance reimbursement and payment of expenses forms and original

receipts. The reasoning system also infers that only the project coordinator can use project funding.

The complete specification can be used by designers to verify if the model complies with desirable security properties. To this intent, the framework supports requirements engineers through the use of *constraints* (Gelfond and Lifschitz 1991). Constraints encode the properties presented in Table 1 into a form that is supported by external ASP solvers. In particular, constraints specify conditions which must not be true in the model. In other words, constraints are formulations of possible inconsistencies. If all constraints are not simultaneously satisfied, weaknesses or vulnerabilities may occur in the actual implementation of the system or in the policies adopted by the organization (Massacci and Zannone 2006). In such situations, it is up to the designer to decide whether or not such failures compromise the system and adopt the adequate countermeasures. This decision depends on several factors such as risk, cost of the solution, compliance with legislation, etc.

Example 5 To verify the compliance of the requirements model with Pro4 and Pro6 in Table 1, Secure Tropos uses the following constraints:

(Pro4) $\leftarrow$ delegate_perm(A,B,S) $\wedge$ not trustChain_perm(A,B,S)
(Pro6) $\leftarrow$ delegate_perm(A,B,S) $\wedge$ not has_perm(A,S)

The analysis reveals the presence of weaknesses in the mission procedure. Pro6 is not satisfied since the adviser has authorized the use of the funding without the consent of the project coordinator. Consequently, the student cannot receive the reimbursement. This is also detected by the reasoning engine in the form of violations of availability requirements. In particular, the student cannot achieve his goal since he has delegated part of it to an actor that does not have the permission to achieve assigned obligations (Pro7).

Other inconsistencies come up since the trust model is not considered in the regulation. Actually, trust relations are implicitly defined in the employment contract that actors draw up with the University. The lack of an explicit trust model makes Pro4 not satisfied.

Inconsistencies of properties might be due to either unspecified requirements or conflicting requirements. Thus, detecting and solving inconsistencies helps requirements engineers to detect implicit and unspecified requirements, understand system vulnerabilities, and identify and evaluate solutions for mitigate vulnerabilities. We note that if such inconsistencies are not resolved, weaknesses or vulnerabilities might occur in the deployed system. For instance, appointing an untrusted actor to achieve critical tasks increases the risk of their failures. Thus, requirements analysis drives the designer in revising and refining the requirements model.

Example 6 The analysis shows how inconsistencies due to the failure of Pro6 can be solved by modifying the mission procedure. In particular, the student shall not depend on the adviser for the use of funding, but he shall ask for authorization directly from the project coordinator.

5 Tool evaluation

The Secure Tropos methodology and ST-tool have been used to model and analyze several industrial case studies (Asnar et al. 2006; Massacci et al. 2005; Massacci and Zannone 2006). In this section we report two of these experiences: compliance by the University of Trento with Italian legislation on Privacy and Data Protection (Massacci et al. 2005) and analysis of a fraud to the detriment of Allied Irish Bank (Massacci and Zannone 2006). An important objective for both case studies was to evaluate the expressiveness of the modeling language and validate the formal framework and analysis. The section also includes an experimental evaluation of the scalability of the proposed automated reasoning techniques.

5.1 Compliance by the University of Trento with Italian legislation on privacy and data protection

In (Massacci et al. 2005) we have used the tool to model and analyze a comprehensive case study on the compliance of the University of Trento with Italian privacy and data protection legislation. The Italian Data Protection Act requires public administrations, which include universities, to set up adequate security and privacy policies. The Act also includes a technical annex that is similar to the ISO-17999 standard. This annex defines the minimal precautionary security measures that should be implemented by administrations, such as authentication and authorization facilities, antivirus protection, as well as data backup and restore mechanisms.

The University has enforced the Data Protection Act through an Internal Privacy Regulation that delegates responsibilities of the data controller (the Chancellor) concerning the processing of personal data to Faculty Deans, Heads of Departments, and Central Directorate Managers. These actors are responsible for fulfilling all obligations relating to personal data processed within the University, with support from the ICT Directorate with regard to the adoption of minimal precautionary security measures for electronic data processing.

Requirements elicitation for the case study was based on the analysis of 300-page documentation, which required three months of work and several interactions with the Information Security Office Manager of the University. The entire requirements model specifies 11 actors, 5 of them were expanded with a total of 90 model elements—77 goals and 13 resources. These 79 elements were linked through a total of 114 links including 25 execution dependencies, 13 permission delegations, and 72 decomposition links. Figures 8(a) and 8(b) present the execution dependency diagram and permission delegation diagram of a fragment of the case study.

Requirements analysis of these models identified revealed a number of pitfalls. For instance, the analysis of procedures and policies adopted by the University pointed out that they do not provide information that is essential for their verification. The most notable omission was the absence of relationships between the Chancellor and the General Director, who is actually responsible for managing the University administration.

Another omission in official documentation was the lack of a definition of the data collection process adopted by the University. This process, and in particular

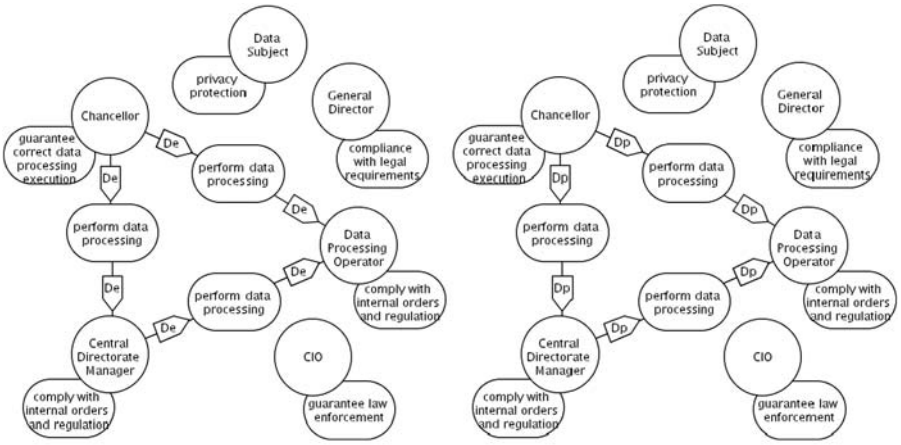


Fig. 8 Requirements model

the need for data subject consent, is considered fundamental in privacy legislation. Moreover, documentation did not identify who is really allowed to perform a specific data processing task. Consequently, it was not possible to analyze availability requirements. This omission also affected the analysis of the need-to-know principle. Indeed, without a precise knowledge about who has the capability to execute a certain data processing task, it was not possible to identify who had actually taken responsibility for its execution. It is worth noting that this problem is not confined to the University of Trento. Rather, it is a generic problem mainly due to the security assessment procedure defined in the annex of the Data Protection Act.

Another issue that was brought up by the analysis was the treatment of manual non-ICT procedures. In fact, such procedures are often incompletely specified. This does not mean that employees do not follow procedures, but rather that such procedures are somehow "embedded" in the knowledge of the organization. Without a precise description of such procedures, the definition of the corresponding authentication procedures and access control policies becomes problematic.

5.2 John Rusnak and the Allied Irish Bank

In (Massacci and Zannone 2006) we demonstrated that the proposed methodology is able to identify the vulnerabilities exploited by a currency trader to defraud the Allied Irish Bank. In the early 90s, the trader John Rusnak gained nearly $500.000 in bonuses for alleged bank profits by exploiting his trader position at Allied Irish Bank (Promontory Financial Group 2003). The analysis we conducted for this case study showed that John Rusnak did not actually hack the IT system, but rather exploited substantial loopholes in the organizational and IT structures that compromised the authenticity and integrity of data upon which a number of decision were taken by Allied Irish Bank's management.

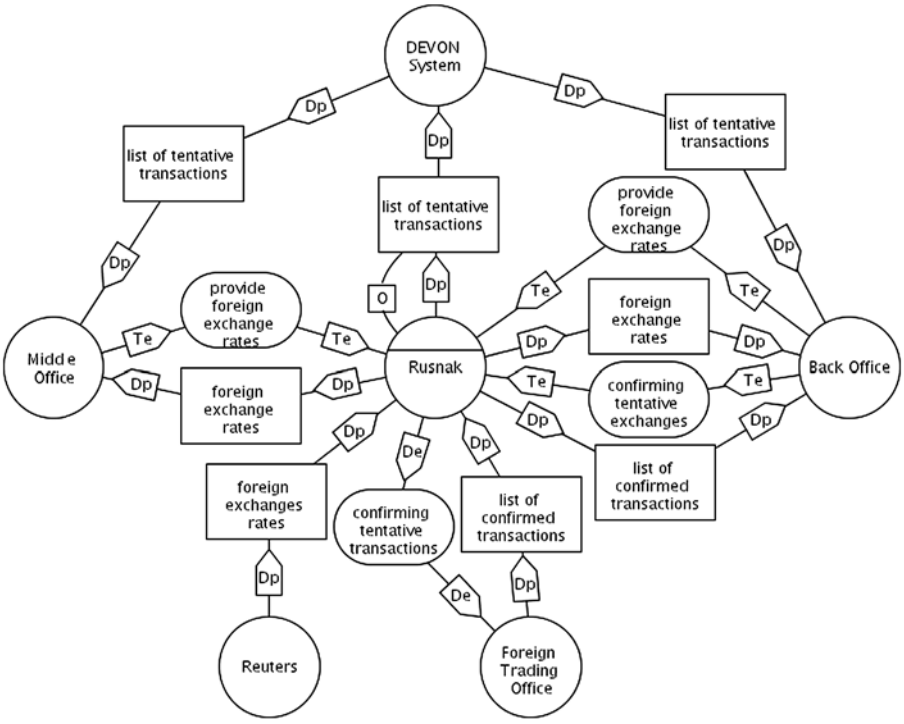


Fig. 9 Bank's organization with Rusnak

This case study was based on the analysis of some official documentation such as the "Ludwig" report (Promontory Financial Group 2003) that was ordered by the Bank to understand Rusnak's fraud and the indictment against Rusnak presented at the Grand Jury for the District of Maryland (US Department of Justice 2002). The entire requirements model includes 15 actors, 6 of them were expanded with a total of 67 model elements—56 goals and 11 resources. These 67 elements were linked through a total of 75 links including 18 execution dependencies, 20 permission delegations, and 22 decomposition links. Figure 9 presents a fragment of the Bank's organizational structure, focusing on the position of Rusnak within the organization.

The fraud designed by Rusnak was based on a number of weaknesses and vulnerabilities affecting the Bank's organizational structure and its IT systems. The lack of integrity protection of foreign exchange rates was one of these vulnerabilities. The Bank developed an architecture where rates were downloaded on Rusnak's machine instead of buying a dedicated Reuters connection for each of its offices. This design solution allowed Rusnak to manipulate exchange rates. This vulnerability was detected by the tool by comparing the Bank's policies with the concrete instantiation of the organization. Actually, bank policy did not permit currency traders to provide foreign exchange rates. Even if this conflict is "visible", it may be ignored by designers due to its nature (since it involves different levels of analysis), and the size of the overall model.

Another vulnerability exploited by Rusnak was the possibility to confirm bogus options. The Bank's policy stated that every trade made by currency traders must be confirmed by the Back Office. However, Rusnak persuaded Back Office employees to let him confirm his own transactions. Such a vulnerability has been automatically detected by the tool by comparing the Bank's organization and policies with their concrete instantiation. Actually, the Back Office was not supposed to permit currency traders to confirm their own transactions. Nonetheless the Back Office employees trusted Rusnak and did not verify the validity of his reported transactions.

Finally, the lack of interaction between the Middle Office and the Back Office introduced further vulnerabilities exploited by Rusnak. The Middle Office computed the value at risk—a category of risk metrics that describe how the market value of an asset is likely to decrease over a certain time period (Jorion 2000)—on tentative trades instead of considering trades confirmed by the Back Office. Thus, Rusnak was able to tamper value at risk by introducing bogus options in the list of tentative transactions. In fact, our analysis did not reveal this vulnerability. The main reason for this was incompleteness of the documentation we worked with. We believe that this vulnerability can be captured by analyzing, for instance, the code of practice for financial markets (Association Cambiste Internationale 2005), which defines the best practices that should be adopted by every bank, and compares these with the actual policies adopted by the Bank.

In summary, the tool allowed one to find automatically some of the errors that were the results of months of investigation by a large team of experts looking at textual documentation manually. For instance, the Promontory Financial Group "have emphasized from the outset that we believed that 30 days was inadequate to render a comprehensive report" (Promontory Financial Group 2003). Even though this analysis was conducted after the attack, it could also have been conducted before. By contrast, the analysis performed by the Promontory Financial Group as well as other kinds of analysis, such as violation and vulnerability analysis (Johnson 2006), can be used to understand the root causes of security incidents, but can only be applied after the attack has occurred. Finally, this case study has confirmed that security issues cannot be addressed only with pure IT solutions. IT systems might be well designed and employ suitable protection mechanisms but be insufficient to fully address security issues. Rather, designers need to look at them from a wider organizational perspective. Only by analyzing the system and the organizational setting wherein it will operate, we can identify some types of weaknesses and vulnerabilities of the system itself.

5.3 Experimental results

Tools may have to provide interactive verification services involving potentially large numbers of clauses. Indeed, the current usage model is that the requirements engineer draws some models, checks for correctness and consistency, revises the model and checks again. Moreover, to perform a more detailed analysis we have also recognized the importance of comparing organizational structure with the concrete (operational) instance of an organization. For instance, this is crucial for capturing security requirements in a domain where a trusted role can be played by an untrusted agent and vice

Table 2 Experimental result

Solver	Cmodels-1			Cmodels-2			Smodels			ASSAT			DLV		
N In	R	Wall	CPU	R	Wall	CPU	R	Wall	CPU	R	Wall	CPU	R	Wall	CPU
0	0	0m13s	<0m1s	0	0m13s	<0m1s	0	0m14s	<0m1s	0	0m14s	<0m1s	0	<0m1s	<0m1s
24	0	0m59s	<0m1s	0	0m58s	<0m1s	0	1m5s	<0m1s	0	1m4s	<0m1s	0	<0m1s	<0m1s
45	0	2m33s	0m2s	0	2m33s	0m1s	0	2m50s	0m2s	0	2m50s	0m1s	0	<0m1s	<0m1s
62	1	0m41s	0m1s	1	0m41s	0m1s	1	0m46s	0m2s	1	0m46s	0m1s	0	<0m1s	<0m1s
113	1	0m47s	0m1s	1	0m47s	0m1s	1	0m54s	0m2s	1	0m54s	0m1s	0	0m2s	<0m1s
166		—			—			—			—		0	0m5s	<0m1s
250		—			—			—			—		0	0m10s	<0m1s
350		—			—			—			—		0	0m25s	<0m1s

versa (Giorgini et al. 2005a). In these settings, where the size of the model depends on the actual size of an organization (in terms of actors), scalability problems may arise.

We have performed several experiments to test the scalability of our approach using different ASP solvers, even when organizational models are instantiated with a growing number of agents playing various roles. In this paper we report the results of the analysis of the case study presented in (Massacci et al. 2005). The experiments were executed on a 2.0 GHz Core Duo processor, 1 GB Ram, running Linux.

Table 2 reports the time used to complete the analysis (Wall) and the CPU time required over models of increasing size. With "0" we mark experiments that completed successfully, while "1" marks those that failed to complete. The results suggest that DLV is far more efficient than other solvers. This is even more evident considering that Wall and CPU times reported in Table 2 do not take into account the time spent by Lparse that is used by Cmodels, Smodels and ASSAT for grounding. The explanation of these results is simple. Engines native to ASP (e.g., the DLV system) are much more efficient than ones that are extended to support ASP, when the number of the instances in the model increases. Moreover, Cmodels, Smodels and ASSAT are not able to find a solution after a certain number of instances due to limits of Lparse.

The results of these experiments suggest that formal analysis based on a DLV solver can handle full-size industrial case studies. After all, enterprises with 250 employees are considered medium-size by the European Union (European Commission Recommendation 2003/361/EC, May 6, 2003).

6 Related work

Several CASE tools have been proposed in the last years to assist developers during requirements elicitation and analysis, but few of them have been extended to cope with security requirements analysis.

The OpenOME tool (Ernst et al. 2006) has been developed to support the i* (Yu 1995) and Non-Functional Requirements (Chung et al. 2000) modeling frameworks, providing requirements engineers with a graphical interface to draw diagrams. These

modeling frameworks treat security requirements as non-functional requirements and model them using softgoals (Liu et al. 2003). OpenOME supports goal analysis (in form of label propagation) to check if the designed system guarantees an appropriate level of security. Liu et al. (2003) extends this approach by offering facilities for threats, vulnerabilities and countermeasures analysis. To support such analysis, this work extends the i* framework with an analysis technique based on Alloy (Jackson 2002). However, such a technique is not integrated into OpenOME. TAOM4E (Perini and Susi 2004) is another tool developed to support the Tropos methodology. Differently from ST-Tool and OpenOME, TAOM4E focuses on the software development process, but it does not offer any facilities for formal requirements analysis.

The GRAIL tool (Darimont et al. 1997) has been developed to support the KAOS methodology (Dardenne et al. 1993). KAOS is a Goal-Oriented Requirements Engineering methodology supporting the whole requirements elaboration process. To cope with security issues, this methodology uses the notion of obstacle to capture exceptional behaviors (van Lamsweerde and Letier 2000) and anti-goal to model intentional obstacles set up by attackers to break security goals (van Lamsweerde 2004). First-order temporal logic is used to model and reason about actor's goals and derive requirements from them. GRAIL provides developers with a graphical interface that supports requirements elicitation and documentation. Moreover, it provides syntax and static semantic checkers at declaration and assertion levels for requirements analysis. However, the support provided by GRAIL is limited to semi-formal specifications. The FAUST toolbox (Rifaut et al. 2003) is designed to integrate formal and semi-formal specifications. It provides tools for the formal analysis of requirements consistency.

The SCR* toolset (Heitmeyer et al. 1998) is a set of tools for developing requirements specifications expressed in the SCR (Software Cost Reduction) tabular notation. The SCR method is a formal method for specifying the requirements of real-time embedded systems. In SCR, the required system behavior is described by mathematical relations. To specify such relations, the method uses the concepts of condition, event, and table. A condition is a predicate defined on one or more variables in the specification. An event represents a change of variable value. Tables specify the value of a variable as a mathematical function defined on conditions and events. To provide a precise and detailed semantics, the SCR method provides a requirements model that represents the system-to-be as a finite state automaton. The toolset includes an editor for defining specifications, a consistency checker for testing the consistency of specification with the formal requirements model, and a simulator for executing the specifications, and a verifier for checking their compliance with application properties.

Compendium (Selvin and Buckingham Shum 2005) is a hypermedia concept mapping tool based on the Issue Based Information System approach and tailored to model problems. It provides extension mechanisms for increasing the expressiveness of the modeling framework. Such mechanisms have been used to support augmentation driven problem analysis (Haley et al. 2005). The objective of the extended tool is to assist developers during the requirements elicitation process (Buckingham Shum et al. 2006). However, Compendium and its extension do not provide facilities for formal requirements analysis.

AUTOFOCUS (Schätz et al. 2002) is a model-based tool designed for the development of distributed and embedded systems. This tool supports system developers

during design phase, offering them a graphical interface for specifying the system-to-be from different views. It also provides formal methods tailored for systems engineering. In particular, it uses consistency criteria on system descriptions and provides formal reasoning techniques for detecting system weaknesses.

The CORAS project has developed a tool-supported methodology for UML-based security risk analysis (den Braber et al. 2003). The tool aids system designers to perform risk analysis and generate documentation reporting results of such analysis. However, the tool itself does not provide novel analysis facilities, but integrates existing techniques such as misuse cases (Sindre and Opdahl 2005) and fault tree analysis (Stamatelatos et al. 2002). Essentially, it provides a methodological approach for integrating different risk analysis approaches for a comprehensive view of the risk management process. The tool allows for storage of the result of the risk analysis process in repositories. These repositories provide reusable experience for future projects.

7 Conclusions

We have presented ST-Tool, a CASE tool designed to support the Secure Tropos methodology for modeling and reasoning about security requirements. The tool has been evaluated by modeling and analyzing real-world, comprehensive case studies with satisfactory results. For instance, the tool was able to identify some of the vulnerabilities that have been exploited by a currency trader to cheat Allied Irish Bank (Massacci and Zannone 2006) and verify the compliance with the University of Trento to Italian legislation on Privacy and Data Protection, leading to the definition and analysis of a ISO-17799-like security management scheme (Massacci et al. 2005).

An important issue left for future research concerns the visualization of the results computed by external ASP solvers. An idea we propose to explore is to construct graphical models that represent the output of solvers so that requirements engineers and stakeholders can directly interact with the formal framework. In particular, our objective is to visually represent violations of security properties. Another open problem for this research is to integrate the security analysis techniques presented in this work with other, complementary ones in order to fully support the analysis and design of secure software systems.

Acknowledgements This work has been partially funded by EU Commission, through the SENSORIA and SERENITY projects, by the FIRB program of MIUR under the TOCAI project, and by the Provincial Authority of Trentino, through the MOSTRO project.

References

Agrawal, R., Kiernan, J., Srikant, R., Xu, Y.: Hippocratic databases. In: Proc. of VLDB'02, pp. 143–154. Kaufmann, Los Altos (2002)

Alpern, B., Schneider, F.B.: Recognizing safety and liveness. Technical Report TR86-727, Cornell University, Computer Science Department (1986)

Anderson, R.: Why cryptosystems fail. Commun. ACM **37**(11), 32–40 (1994)

Asnar, Y., Bonato, R., Bryl, V., Compagna, L., Dolinar, K., Giorgini, P., Holtmanns, S., Klobucar, T., Lanzi, P., Latanicki, J., Massacci, F., Meduri, V., Porekar, J., Riccucci, C., Saidane, A., Seguran, M., Yautsiukhin, A., Zannone, N.: Security and privacy requirements at organizational level. Research report A1.D2.1, SERENITY consortium (2006)

Association Cambiste Internationale: The model code: the international code of conduct and practice for the financial markets (2005). http://www.aciforex.com/market/July05_ModelCode.pdf

Avizienis, A., Laprie, J.-C., Randell, B., Landwehr, C.E.: Basic concepts and taxonomy of dependable and secure computing. IEEE Trans. Dependable Secur. Comput. **1**(1), 11–33 (2004)

Basin, D., Doser, J., Lodderstedt, T.: Model driven security: from UML models to access control infrastructures. ACM Trans. Softw. Eng. Methodol. **15**(1), 39–91 (2006)

Bauer, L., Ligatti, J., Walker, D.: More enforceable security policies. In: Proc. of FCS'02 (2002)

Bresciani, P., Giorgini, P., Giunchiglia, F., Mylopoulos, J., Perini, A.: TROPOS: an agent-oriented software development methodology. J. Auton. Agents Multi-Agent Syst. **8**(3), 203–236 (2004)

Buckingham Shum, S.J., Selvin, A.M., Sierhuis, M., Conklin, J., Haley, C.B., Nuseibeh, B.: Hypermedia support for argumentation-based rationale: 15 years on from gIBIS and QOC. In: Rationale Management in Software Engineering, pp. 105–126. Springer, Berlin (2006)

Chung, L.K., Nixon, B.A., Yu, E., Mylopoulos, J.: Non-Functional Requirements in Software Engineering. Kluwer, Dordrecht (2000)

Dardenne, A., van Lamsweerde, A., Fickas, S.: Goal-directed requirements acquisition. Sci. Comput. Prog. **20**, 3–50 (1993)

Darimont, R., Delor, E., Massonet, P., van Lamsweerde, A.: GRAIL/KAOS: an environment for goal-driven requirements engineering. In: Proc. of ICSE'97, pp. 612–613. ACM Press, New York (1997)

De Landtsheer, R., van Lamsweerde, A.: Reasoning about confidentiality at requirements engineering time. In: Proc. of ESEC/FSE'05, pp. 41–49. ACM Press, New York (2005)

den Braber, F., Dimitrakos, T., Gran, B.A., Lund, M.S., Stølen, K., Aagedal, J.Ø.: The CORAS methodology: model-based risk assessment using UML and UP. In: UML and the Unified Process, pp. 332–357. Idea Group Publishing, New York (2003)

Ernst, N.A., Yu, Y., Mylopoulos, J.: Visualizing non-functional requirements. In: Proc. of REV'06, p. 2. IEEE Press, New York (2006)

Fickas, S., Nagarajan, P.: Critiquing software specifications. IEEE Softw. **5**(6), 37–47 (1988)

Fuxman, A., Liu, L., Mylopoulos, J., Pistore, M., Roveri, M., Traverso, P.: Specifying and analyzing early requirements in tropos. Requir. Eng. J. **9**(2), 132–150 (2004)

Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In: Proc. of ICLP'88, pp. 1070–1080. MIT Press, Cambridge (1988)

Gelfond, M., Lifschitz, V.: Classical negation in logic programs and disjunctive databases. New Gener. Comput. **9**(3/4), 365–386 (1991)

Germeau, F., Leduc, G.: Model-based design and verification of security protocols using LOTOS. In: Proc. of the DIMACS Workshop on Design and Formal Verification of Security Protocols (1997)

Giorgini, P., Massacci, F., Mylopoulos, J., Zannone, N.: Modelling social and individual trust in requirements engineering methodologies. In: Proc. of iTrust'05. Lecture Notes in Computer Science, vol. 3477, pp. 161–176. Springer, Berlin (2005a)

Giorgini, P., Massacci, F., Zannone, N.: Security and trust requirements engineering. In: FOSAD 2004/2005. Lecture Notes in Computer Science, vol. 3655, pp. 237–272. Springer, Berlin (2005b)

Giorgini, P., Massacci, F., Mylopoulos, J., Zannone, N.: Requirements engineering for trust management: model, methodology, and reasoning. Int. J. Inform. Sec. **5**(4), 257–274 (2006)

Gravell, A.M., Henderson, P.: Executing formal specifications need not be harmful. IEE/BCS Softw. Eng. J. **11**(2), 104–110 (1996)

Haley, C.B., Moffett, J., Laney, R., Nuseibeh, B.: Arguing security: validating security requirements using structured argumentation. In: Proc. of SREIS'05 (2005)

Heitmeyer, C.L., Kirby, J., Labaw, B.G., Bharadwaj, R.: SCR*: A toolset for specifying and analyzing software requirements. In: Proc. of CAV'98, pp. 526–531. Springer, Berlin (1998)

House of Lords, P.: Prince Jefri Bolkiah vs KPMG. 1 All ER 517 (1999). Available on www.parliament.the-stationeryoffice.co.uk

Jackson, D.: Alloy: a lightweight object modelling notation. ACM Trans. Softw. Eng. Methodol. **11**(2), 256–290 (2002)

Johnson, C.W.: V2: using violation and vulnerability analysis to understand the root causes of complex security incidents. Submitted to ACM Trans. Inf. Syst. Secur. (2006)

Jorion, P.: Value-at-Risk: The New Benchmark for Managing Financial Risk. McGraw–Hill, New York (2000)

Jürjens, J.: Secure Systems Development with UML. Springer, Berlin (2004)
Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S., Scarcello, F.: The DLV system for knowledge representation and reasoning. ACM Trans. Comput. Log. **7**(3), 499–562 (2006)
Lierler, Y.: Disjunctive answer set programming via satisfiability. In: Proc. of the 3rd Int. Workshop on Answer Set Prog.: Adv. in Theory and Implementation, CEUR Workshop Proceedings. CEUR-WS.org, vol. 142 (2005)
Lin, F., Zhao, Y.: ASSAT: computing answer sets of a logic program by SAT solvers. In: Proc. of the 18th Nat. Conf. on Artif. Intell, pp. 112–117. AAAI Press, Menlo Park (2002)
Liu, L., Yu, E.S.K., Mylopoulos, J.: Security and privacy requirements analysis within a social setting. In: Proc. of RE'03, pp. 151–161. IEEE Press, New York (2003)
Maiden, N., Sutcliffe, A.: Exploiting reusable specifications through analogy. CACM **35**(4), 55–64 (1992)
Massacci, F., Prest, M., Zannone, N.: Using a security requirements engineering methodology in practice: the compliance with the Italian data protection legislation. Comput. Stand. Interfaces **27**(5), 445–455 (2005)
Massacci, F., Zannone, N.: 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)
McDermott, J., Fox, C.: Using abuse case models for security requirements analysis. In: Proc. of ACSAC'99, pp. 55–66. IEEE Press, New York (1999)
National Security Agency: Information Assurance Technical Framework (IATF). Release 3.1 (2002)
Niemelä, I., Simons, P.: Efficient implementation of the well-founded and stable model semantics. In: Proc. of JICSLP'96, pp. 289–303. MIT Press, Cambridge (1996)
Niemelä, I., Simons, P., Syrjänen, T.: Smodels: a system for answer set programming. In: Proc. of the 8th Int. Workshop on Non-Monotonic Reas. (2000)
Nuseibeh, B., Easterbrook, S.: Requirements engineering: a roadmap. In: Proc. of ICSE'00, pp. 35–46. ACM Press, New York (2000)
Onabajo, A., Jahnke, J.H.: Modeling and reasoning for confidentiality requirements in software development. In: Proc. of ECBS'06, pp. 460–467. IEEE Press, New York (2006)
Perini, A., Susi, A.: Developing tools for agent-oriented visual modeling. In: Proc. of MATES'04. Lecture Notes in Computer Science, vol. 3187, pp. 169–182. Springer, Berlin (2004)
Promontory Financial Group, Wachtell, Lipton, Rosen, Katz: Report to the Board and Directors of Allied Irish Bank P.L.C., Allfirst Financial Inc., and Allfirst Bank Concerning Currency Trading Losses (2003)
Rifaut, A., Massonet, P., Molderez, J.-F., Ponsard, C., Stadnik, P., van Lamsweerde, A., Hung, T.V.: FAUST: formal analysis using specification tools. In: Proc. of RE'03, p. 350. IEEE Press, New York (2003)
Saltzer, J.H., Schroeder, M.D.: The protection of information in computer systems. Proc. IEEE **63**(9), 1278–1308 (1975)
Schätz, B., Pretschner, A., Huber, F., Philipps, J.: Model-based development of embedded systems. In: Proc. of OOIS'02. Lecture Notes in Computer Science, vol. 2426, pp. 298–312. Springer, Berlin (2002)
Schneider, F.B.: Decomposing properties into safety and liveness. Technical Report TR87-874, Cornell University, Computer Science Department (1987)
Selvin, A.M., Buckingham Shum, S.J.: Hypermedia as a productivity tool for doctoral research. New Rev. Hypermedia Multimedia **11**(1), 91–101 (2005)
Sindre, G., Opdahl, A.L.: Eliciting security requirements with misuse cases. Requir. Eng. J. **10**(1), 34–44 (2005)
Stamatelatos, M., Vesely, W., Dugan, J., Fragola, J., Minarick, J., Railsback, J.: Fault Tree Handbook with Aerospace Applications. NASA, Washington (2002)
US Department of Justice: United States of America v. John M. Rusnak. SMS/SD/USAO #2002R02005. (2002). http://www.usdoj.gov/dag/cftf/chargingdocs/allfirst.pdf
van Gelder, A.: The alternating fixpoint of logic programs with negation. In: Proc. of PODS'89, pp. 1–10. ACM Press, New York (1989)
van Lamsweerde, A.: Elaborating security requirements by construction of intentional anti-models. In: Proc. of ICSE'04, pp. 148–157. IEEE Press, New York (2004)
van Lamsweerde, A., Letier, E.: Handling obstacles in goal-oriented requirements engineering. IEEE Trans. Softw. Eng. **26**(10), 978–1005 (2000)
Yu, E.: Modelling strategic relationships for process reengineering. PhD thesis, University of Toronto (1995)