Security Requirements Engineering: the SI* Modeling Language and the Secure Tropos Methodology

Fabio Massacci, John Mylopoulos and Nicola Zannone

Abstract Security Requirements Engineering is an emerging field which lies at the crossroads of Security and Software Engineering. Much research has focused on this field in recent years, spurred by the realization that security must be dealt with in the earliest phases of the software development process as these phases cover a broader organizational perspective. Agent-oriented methodologies have proved to be especially useful in this setting as they support the modeling of the social context in which the system-to-be will operate. In our previous work, we proposed the SI* modeling language to deal with security and trust, and the Secure Tropos methodology for designing secure software systems. Since then, both have been revised and refined in light of experience gained from their application to several industry case studies. This chapter presents the consolidated versions of the SI* modeling language and the Secure Tropos methodology and recounts our experiences, explaining the practical and theoretical reasons behind each consolidation step.

1 Introduction

Security is widely recognized as one of the main challenges in developing software. Contrary to what may be expected from the strong emphasis given to buffer overflow or other software errors in the popular press, the analysis of security incidents and frauds [2, 7, 24, 42] has revealed that security is compromised most often not by breaking protection mechanisms such as encryption or security protocols; the major cause is the exploitation of loopholes at the interface between the organization

Fabio Massacci
University of Trento, e-mail: massacci@unitn.it

John Mylopoulos
University of Trento, e-mail: jm@disi.unitn.it

Nicola Zannone (Corresponding author)
Eindhoven University of Technology , e-mail: n.zannone@tue.nl

and the IT system, and, consequently, in the security policies adopted by the organization as whole (as opposed to measures adopted by the IT system only). The IT system might be well designed and employ suitable security solutions but be insufficient to address security issues because only some of the organization processes are captured in the process under the control of the IT system. Security should thus be considered during the overall system development process including the analysis of the organization in which the system will at end operate.

In contrast, we only find a well-developed literature for software engineering of system-oriented aspects of security (e.g., by extending UML) [8, 19, 26, 38, 43, 49], as well as early requirements models addressing security concerns of IT systems [30, 40, 51, 53]. For instance, Jürjens introduces security tagging to represent the need of a security-protection mechanism in UML collaboration diagrams [26]. The evidence, such as the reports of the Association of Certified Fraud Examiners [7], suggests that for security engineering, it is also necessary to model the organization and the social relationships among all actors involved in the system.

This issue has been only partly addressed by early requirements engineering approaches [30, 51]. These approaches support the modeling of attackers (both internal and external) along with their objectives as well as the representation of design decisions that can contribute to a security goal. However, they usually lack fundamental concepts needed to talk about security within an organization. Most early requirements engineering approaches do not support concepts such as ownership and trust, which are at the very foundation of all security concerns. Ownership indicates that there is something to protect: if people do not own data rights, privacy rights, or physical property, security would be a meaningless concept. Trust represents the willingness to accept risks based on positive expectations about the behavior of another actor [37]. The absence of constructs for capturing trust affects decisions about the security measures imposed on the system, which might be excessive in some cases or inadequate in others. The presence of trust (or lack thereof) allows designers to economize on information processing and protection mechanisms.

What is missing is a methodology for describing organizations and their operational procedures, and then deriving security policies and mechanisms from the requirements analysis process in the style of Model-Driven Development. If we look at the requirement refinement process of many research papers, we find out a gap between security measures and the requirements of the entire IT system: we have a system with no security features consisting of high-level functionality, and then the next refinement shows encryption, access control, authentication and other security solutions. At the organizational level we miss what has been already achieved at the system level by Basin et al. [8] and, namely, the ability to derive access control policies from UML-like security specifications.

In our previous works [21, 22], we have proposed the SI* modeling language and the Secure Tropos methodology to address the problem of modeling and analyzing security requirements at the organizational level. Since then, SI* and Secure Tropos have been adopted in several European, national, and local projects for the analysis of early organizational and security requirements of different application domains [4, 5, 6, 15, 23, 34, 36]. For the sake of compactness, we only report the

This case study analyzes a health care system where patients are equipped with intelligent devices which monitor their health status 24 hours a day and transmit the information to a health care center. Several privacy issues arise from this case study, which have been analyzed combining requirement engineering and a legal perspective [15].

(a) A Smart Item Infrastructure for e-Health

The aim of this case study is to bring security issues related to e-Government to light by analyzing a fiscal portal and the on-line services it offers to citizens and companies. In this context, the focus is on the identification and authentication of the tax-payer and anonymous consulting of the tax information database [23].

(b) A TaxPayer Portal

This case study addresses a loan process along with the complex activities performed by a bank for actually granting it. Those included the verification of the identity and financial credentials of clients and other potentially outsourced activity such as rating agency evaluations beside a number of separation of duty constraints [46].

(c) Loan Origination Process

Air Traffic Management (ATM) systems are required to ensure the safe and efficient movement of aircraft during all phases of operation. This case study focuses on resectorization and partial airspace delegation processes due to an unplanned increase of air traffic with particular attention to dependability aspects of the system [5, 6].

(d) Partial Airspace Delegation Process:

Fig. 1 SERENITY Case Studies

case studies from the industry-lead EU SERENITY project[1] ranging from Smart Item and Air Traffic Management to e-Business and e-Government (briefly summarized in Fig. 1). The application of the language and of the methodology to those case studies has allowed us to evaluate their expressiveness and usability and prove their applicability in industry. The attempt to capture and analyze the issues raised by the case studies has pointed out a number of drawbacks in our initial proposal, which have demanded for a revision of SI* and Secure Tropos.

Bringing SI* up to capturing challenges of cases studies has required to

- drop some constructs from the ancestor i* and our own earlier proposal to improve the readability and manageability of the framework by industry partners;
- add new constructs to express a number of modeling situations that were frequent but cumbersome, if not impossible, to express with existing constructs;
- rename some concepts to make them closer to the first intuition that an industry user might have of them.

The use of the Secure Tropos methodology in the projects has also revealed other difficulties. The final goal was to have industry partners drawing SI* models by themselves. Initially, they were not confident with the assigned task. Additional methodological support was needed to assist industry partners during requirements elicitation, modeling and analysis. This proved to be a challenging task that required to re-think the original simple process and develop tools. The attempt to meet the challenge has required to

[1] EU-IST-IP 6th Framework Programme – SERENITY 27587 – http://www.serenity-project.org

- restructure the modeling process, reordering modeling actions to meet the steps of the (informal) security engineering process in current industry practices;
- add new phases of the requirements engineering process to support the verification of security requirements and the mitigation of vulnerabilities;
- provide methodological support to bridge the gap between specifications in natural language and SI* models along the line of the VOLERE methodology [44].

This chapter presents a comprehensive and updated description of SI* and Secure Tropos together with the consolidation steps made to capture and analyze the issues raised from their application to industry case studies. Pieces of this work have appeared in different papers. The SI* modeling language along with a formal framework for security requirements analysis was published in [22] and further refined in [33]. Security patterns are described in [15]. The description of a tool that supports designers in requirements modeling and analysis is presented in [32]. The application of a lightweight method for the transformation of requirements specications expressed in natural language into semi-structured specifications to support designers during requirements elicitation is described in [27]. In addition, we have improved the requirements analysis process by providing a procedure for instantiating requirements specified at the organizational level for a more accurate analysis; also how the result of the analysis can be used to drive designers in the application of security patterns for security mitigation. The overall framework is the result of the ambitious goal of bringing a security requirements engineering methodology to industry and having people from industry feel self-confident in using it. This is only the first step as we mostly interacted with people active or somehow participating in research; targeting practitioners is next step.

The reimander of the chapter is organized as follows. The next section presents the consolidated version of SI* and Section 3 presents Secure Tropos. Section 4 describes their evolution based on the application to case studies. Section 5 discusses related work, and Section 6 concludes with some directions for future work.

2 The SI* Modeling Language

The SI* modeling language evolved from the i* modeling language [54] that employs the notions of actor, goal, task, resource, and social dependency between actors to represent the functional design of the system. The i* framework models security requirements as objects of the modeling activity [31], but it lacks constructs to represent them at the meta-level. To model security, trust, and privacy aspects of a socio-technical system, we need to capture the complex web of relations among the different actors participating to the system as well as distinguish between actors who want the fulfillment of a goal, from actors who have the capabilities to do it, and, last but not least, actors who are entitled to do it. This section presents a comprehensive description of the concepts offered by SI*. The concepts are motivated and illustrated by examples drawn on the case studies presented in Fig. 1.

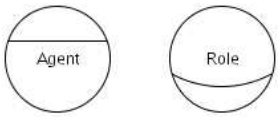


Fig. 2 SI* graphical representations of agents and roles

Agents, Roles and Hierarchical Relations The basic idea behind i* was the concept of actor: an active entity that has strategic goals and performs actions to achieve them. We only allow two different constructs: an *agent* is an active entity with concrete manifestations and is used to model humans as well as software agents and organizations; a *role* is the abstract characterization of the behavior of an active entity within some context. In a loose comparison with OO models, roles are classes and agents are instances. Fig. 2 shows the graphical notation of agents and roles. For the sake of simplicity, we will use the term "actor" throughout the chapter to refer to both agents and roles when it is not necessary to distinguish them. The relation between agents and roles is described by the *play* relation: an agent can play a role.

To capture complex organizational structures, we have identified three additional relations among roles. The first two relations have also a natural counter-part in OO models and are the relations of *component* and *specialization*. *is-part-of* is a relation between two roles[2] and is used to "decompose" complex roles into subcomponents such as the internal structure of an organization or the logical sub-components of a software agent. *is-a* is a relation between two roles and indicates that a role refers to more specialized activities of another role. The specialized role inherits all properties of the generalized role. These relations are employed to understand the organization structure and build role specialization hierarchies, respectively. An example referring to the ATM scenario shows the use of relations *is-part-of*, *is-a*, and *play*.

Example 1. Each Air Traffic Control Center (ACC) divides its airspace into several adjacent volumes, called *sectors*. For instance, ACC-A divides its airspace into 3 sectors (e.g., 1-A, 2-A, 3-A). Each sector is managed by a team consisting of an Executive Controller (EC) (e.g., Edison is the 1-A EC), and a Planning Controller (PC) (e.g., Paul is the 1-A PC). Each team is responsible for the safety of overflight aircraft in its sector. This fragment of the ATM scenario is presented in Fig. 3.

The third relation has no natural counterpart in OO modeling languages. It has only sense when modeling organizations and humans because it deals with a basic primitive concept: the power of man upon his fellows. *supervise* is a relation between two roles and indicates the line of authority between them: a role (the *supervisor*) is responsible for the behavior of the other role (the *subordinate*). The supervisor has the power and responsibility to evaluate and review subordinates' work and monitor their behavior. This construct is used to build the role supervision hierarchy (also called organization chart or organization hierarchy) which describes the flow of authority in an organization through the hierarchy.

[2] We also overload the term and use it for relations between agents with a similar semantics.

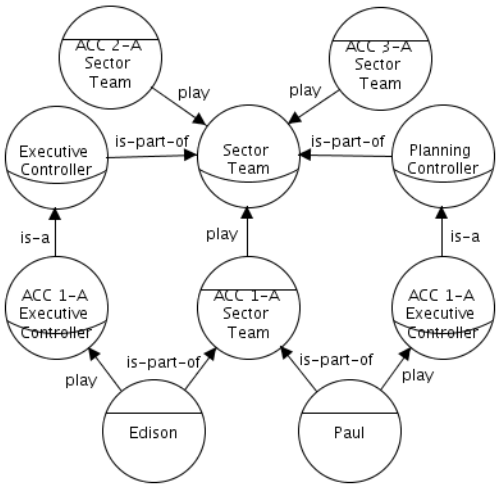
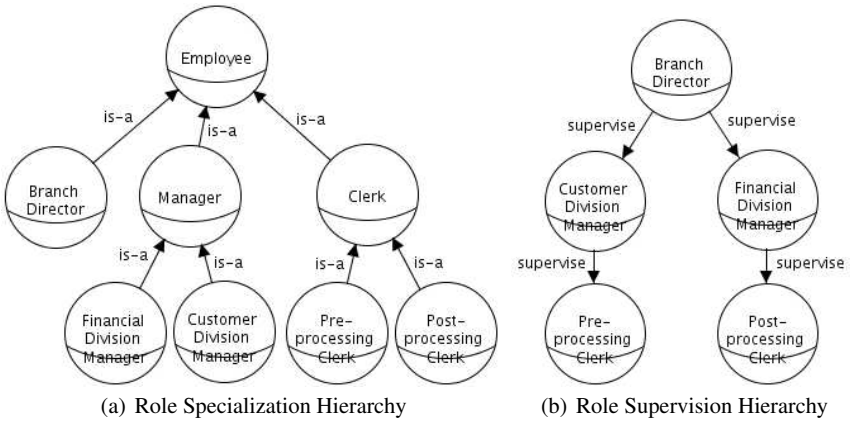


Fig. 3 Play and is-part-of



(a) Role Specialization Hierarchy (b) Role Supervision Hierarchy

Fig. 4 Role Hierarchies

Example 2. The director of a bank is responsible for the correct execution of the loan originator process. Managers and clerks (e.g., pre-processing and post-processing clerks) are appointed by the bank to execute the loan originator process under the supervision of the bank director. If the loan originator process is not executed in compliance with bank policies, the bank director is personally liable. Thereby, he has to take the measures necessary to evaluate and review the work done by subordinates and monitor their behavior. Fig. 4 summarizes the roles presented above, pointing out the differences between the role specialization hierarchy (Fig. 4(a)) and the role supervision hierarchy (Fig. 4(b)).

Fig. 5 SI* graphical representation of goals, softgoals, and resources

Objectives, Entitlements and Capabilities Traditional goal-oriented methodologies, such as KAOS [17] or Tropos [10] and the same i*, have a clear cut notion of *goal*: a state of affairs of an actor that the design should possibly fulfill. Here we must broaden this notion because designers must be able to model situations in which the actors who are capable of fulfilling a goal are different from the ones who are entitled to do it and both are different from the actors who want the goal fulfilled. Accordingly, a *goal* is a state of affairs whose realization is desired by some actor, can be realized by some (possibly different) actor, or is controlled by some (possibly different) actor. In the same way as i*, SI* differentiates between hard goals (simply goals hereafter) and *softgoals*. The latter have no clear criteria for deciding whether they are satisfied or not [14]. They correspond to qualitative properties of the system. A *resource* is a physical or an informational entity without intentionality. The graphical notation of goals, softgoals, and resources is presented in Fig. 5.

Objectives, entitlements and capabilities of actors are modeled through relations between actors and a goal or a resource. *Request* denotes the objectives of actors: an actor (the *requester*) wants a goal achieved or a resource delivered. *Own* denotes the entitlements of actors: an actor (the *owner*) has full authority concerning access to a resource and disposition over the achievement of a goal, that is, the owner of a goal is the one who can decide who can achieve the goal and how it can be achieved. *Provide* denotes the capabilities of actors: an actor (the *provider*) has the ability and knowledge necessary to achieve a goal or furnish a resource. In graphical diagrams, above relations are represented as edges between an actor and a goal or a resource labeled with **R**, **O** and **P**, respectively.

Example 3. In the loan origination process scenario, a customer is the owner of his personal data. He is the one who can decide who can access his data and for which purpose. Post-processing clerks are appointed by the bank to verify customer financial credentials. They need to access customer information to achieve assigned duties. Post-processing clerks, thus, act as data requesters. The clerks however do not directly interact with customers, but retrieve data from the bank IT system. Actually, the bank IT system stores customer information and makes it available to those employees that need such information to achieve assigned duties. Accordingly, the bank IT system acts as the data provider.

Decomposition and Strategic Relations Goals of the same actor or of different actors can be related to one another in many ways. Building upon Tropos, we have identified the following three relations: *AND/OR decomposition*, *means-end*, and *contribution*. AND/OR decomposition refines goals into subgoals, modeling a finer goal structure. In essence, AND-decomposition refines a goal in subparts that must be achieved in order to achieve the goal, while OR-decomposition defines design

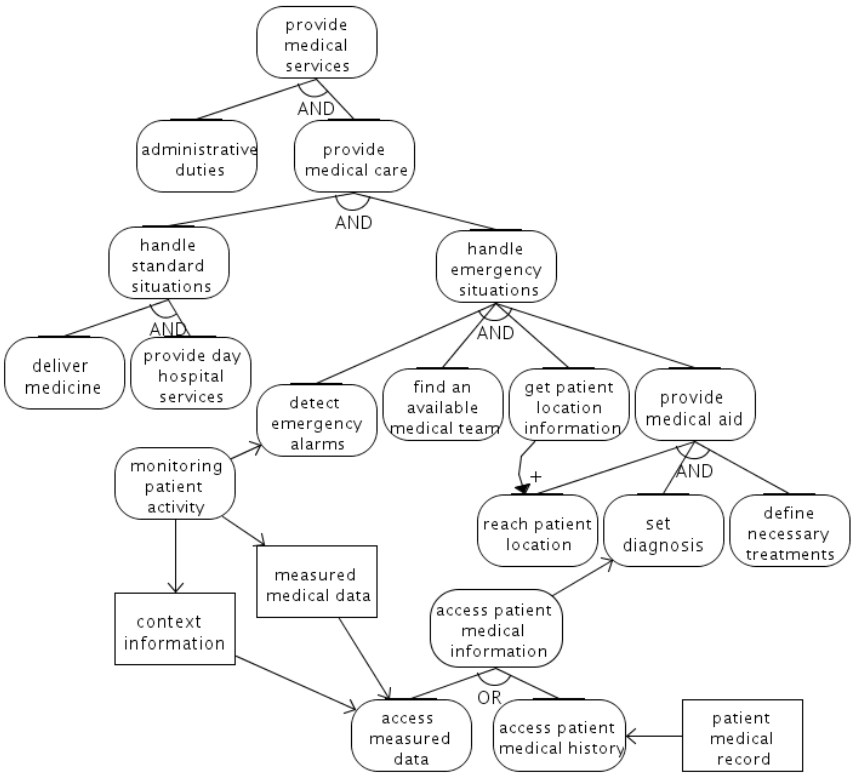


Fig. 6 Goal diagram

alternatives to achieve a goal. Means-end relations identify the goals that provide means to achieve a goal and the resources needed and produced by a goal. Such goals and resources, however, can be not under the control of the actor nor the actor can be able to achieve them. Contribution relations model the impact of the achievement of goals on the system. Normally, one uses the term contribution when the relation between goals is not the consequence of a deliberative planning but rather the result of side-effects. The impact can be positive or negative and is graphically represented as edges labeled with "+" and "−", respectively.

Example 4. In the Smart Item scenario, the main goal of the HCC is to provide medical services to its patients. As shown in Fig. 6, this goal can be decomposed in subgoals, such as the provision of medical care and administrative duties. The HCC shall provide medical care in both standard and emergency situations. These subgoals can be further decomposed. For instance, providing medical aid consists of reaching the patient location, setting a diagnosis, and defining the necessary treatments. To set a diagnosis, a doctor needs access to patient medical information (e.g., measured medical data or patient medical history). However, the doctor may not be entitled to access them. This issue is captured in the model though the use of a

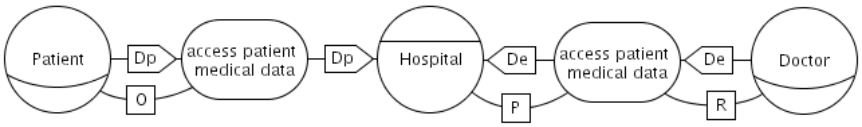


Fig. 7 Execution Dependency and Permission Delegation

means-end relation. Similarly, resources are linked to the goals whose achievement needs or produces them using means-end relations. In Fig. 6, monitoring patient activity results in context information and measured medical data. These informational entities are needed to achieve the goal of accessing measured data.

SI* supports the notions of execution dependency and permission delegation to model the transfer of objectives and entitlements from an actor to another. *Execution dependency* indicates that one actor (the *depender*) appoints another actor (the *dependee*) to achieve a goal or furnish a resource (the *dependum*). As consequence of execution dependency, the dependee wants the achievement of the goal or the delivery of the resource. Hereafter assigned objectives are also called responsibilities as it is assumed that the dependee commits to achieving the goal or delivering the resource. *Permission delegation* indicates that one actor (the *delegator*) authorizes another actor (the *delegatee*) to achieve a goal or deliver a resource (the *delegatum*). As consequence of permission delegation, the delegatee is authorized to achieve the goal or deliver the resource. As suggested by Li at al. [29] for their delegation logic, delegations and dependencies have depth which represents the number of re-delegation steps allowed. The framework also allows the specification of conditions to control (re-)delegation [21]. Execution dependencies and permission delegations are graphically represented as edges respectively labeled with **De** and **Dp**.

Example 5. The scenario of Example 4 reveals that the HCC shall seek the consent of patients for processing their medical data. It also points out that doctors depend on the HCC for accessing such data. These relations are captured in Fig. 7.

Depending on actors for achieving a goal (or delegating the permission to do it) makes the depender (the delegator) vulnerable. Actually, the dependee may not achieve the assigned responsibilities even if he has committed it. Similarly, the delegator has no warranty that the delegatee does not misuse the granted permission. The need to capture these issues has spurred us to separate the concepts of trust and delegation. This separation allows the modeling of systems where some actor must delegate permission or assign responsibilities to untrusted actors (e.g., coercive or blind delegations [13]). Similarly to delegation and dependency, trust is a ternary relation representing the expectations that an actor (the *trustor*) has concerning the behavior of another actor (the *trustee*). The object (i.e., a goal or a resource) around which a trust relation centers is called *trustum*. Also in this case, it is necessary to have a distinction between trust in managing permission and trust in managing execution. *Trust of execution* models the trustor's expectations concerning the ability and dependability of the trustee in achieving a goal or delivering a resource. By

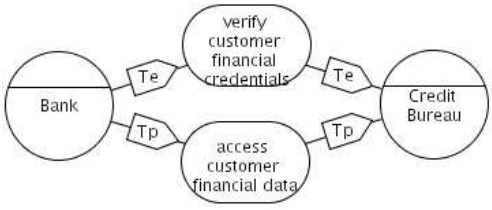


Fig. 8 Trust

trusting in execution, the trustor is sure that the trustee accomplishes the trustum. *Trust of permission* models the trustor's expectations that the trustee does not misuse a goal or a resource. By trusting in permission, the trustor is sure that the trustee does not abuse the (possible) received permission for accomplishing a purpose different from the one for which the permission has been granted. These relations are represented as edges respectively labeled with **Te** and **Tp**.

Example 6. Trust concerns are particular evident in the e-Business scenario where the bank outsources the verification of customer financial credentials to the Credit Bureau. The long-term collaboration between these two financial entities has given evidence to the bank that the Credit Bureau is able to achieve the verification activities and uses customer information only for the assigned duties. This scenario is shown in Fig. 8.

3 The Secure Tropos Methodology

Secure Tropos enhances Tropos [10], providing the support necessary to model and analyze security requirements. This section presents Secure Tropos along the phases supported by the methodology and the requirements analysis process.

3.1 Requirements Analysis Phases

Requirements analysis is composed of two main phases: *Early Requirements* and *Late Requirements* analysis. Both phases share the same conceptual and methodological approach. Thus, most of the ideas introduced for early requirements can be reused for late requirements.

- *Early Requirements* phase concerns the understanding of the application domain by studying the organizational context in which the system-to-be will eventually operate. During this phase, domain stakeholders are identified and modeled as agents and roles together with their objectives, entitlements, capabilities, and their interrelations. The requirements model is then evaluated against a number

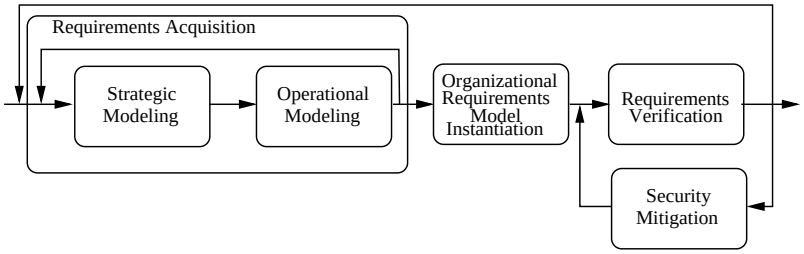


Fig. 9 Requirements Analysis Process

of security and privacy properties. From this analysis, the model is incrementally revised and extended to assure an adequate level of security by taking advantage of the use of security patterns. This process makes it possible to understand the *why*, besides the *what* and *how*, of system functionality and security solutions and, last but not least, verify if the system design matches stakeholder needs.

- *Late requirements* phase concerns the analysis of the system-to-be within its operational environment. The conceptual model is extended including new actor(s) representing the system-to-be together with relations between the system-to-be and the other actors part of the environment. During this phase, the focus of the analysis is on the interface between organizational procedures and the system that support them, besides the analysis of the system-to-be itself.

3.2 Requirements Analysis Process

This section presents the requirements analysis process underlying Secure Tropos (Fig. 9). This is an iterative process composed of the following conceptual phases:

1. *Requirements Acquisition Process*, in which modeling activities are used to build security requirements models;
2. *Organizational Requirements Model Instantiation*, in which the organizational requirements model is instantiated;
3. *Requirements Verification*, in which the compliance of the requirements model with a number of security properties is verified;
4. *Security Mitigation*, in which the requirements model is revised by introducing security measures to cope with the violation of security properties.

Requirements Acquisition Process Secure Tropos adopts SI* to represent the design of socio-technical systems. Various activities contribute to the acquisition of the requirements model and to its refinement into subsequent models. These activities can be grouped in two main classes: *strategic modeling* and *operational modeling*. Strategic modeling is intended to identify and model domain stakeholders and system actors along with the social relations among them. Secure Tropos proposes the following modeling activities:

- *Actor modeling* consists of identifying and analyzing agents and roles within the socio-technical system. Agents are also described in terms of the roles they play. This modeling activity produces the *actor diagram* that represents the roles and agents participating to the system along with their objectives, entitlements, capabilities, and agent-role assignments as well as role specialization hierarchies and the structure of the socio-technical system in terms of component relations.
- *Social Modeling* consists of identifying the social relations among agents and roles. It is composed of *trust modeling* and *supervision modeling*. Trust modeling consists of modeling the expectations that actors have concerning the dependability and fair behavior of other actors. Such expectations are modeled using trust of execution and trust of permission relations. This modeling activity produces the *trust diagram* that enriches the actor diagram with the trust network. Supervision modeling consists of modeling the structure of the organization in terms of supervision relations. This modeling activity produces the *organization chart*.
- *Goal modeling* proceeds in order to enrich the requirements model with further details. Specifically, goal modeling rests on the analysis of goals and resources from the perspective of single actors using AND/OR decompositions, means-end relations, and contribution relations. A graphical representation of goal modeling is given through the *goal diagram* that appears as a balloon within which goals and resources of a specific actor are analyzed.

Operational modeling attempts to capture the operational aspects of socio-technical systems. It is compounded of the following modeling activities:

- *Execution Dependency modeling* consists of identifying actors who assign the responsibility of achieving a goal or furnishing a resource to other actors. Assignments of responsibilities are modeled through execution dependency relations. The outcome of this activity is the *execution dependency diagram* that enriches the requirements model with the execution dependency network.
- *Permission Delegation modeling* consists of identifying actors who authorize other actors to achieve goals or deliver resources. Such transfers of authority are modeled using permission delegation relations. This modeling activity produces the *permission delegation diagram* that enriches the requirements model with the permission delegation network.

The requirements acquisition process (Fig. 10) is an iterative process in which the modeling activities capture different views of the requirements model. The basic idea is to analyze objectives on behalf of different actors through a procedure that leads to their secure achievement. The process starts with the actor modeling activity in which the relevant actors are elicited and modeled together with their objectives, entitlements, and capabilities. An actor may not have the capabilities to fully achieve his objectives, so he can either appoint another actor to fulfill them entirely or decompose them and assign part of them to other actors. The assignment of responsibilities and permissions is driven by the expected behavior of other actors and the organizational structure. The actor diagram thus is used as input for social modeling or goal modeling.

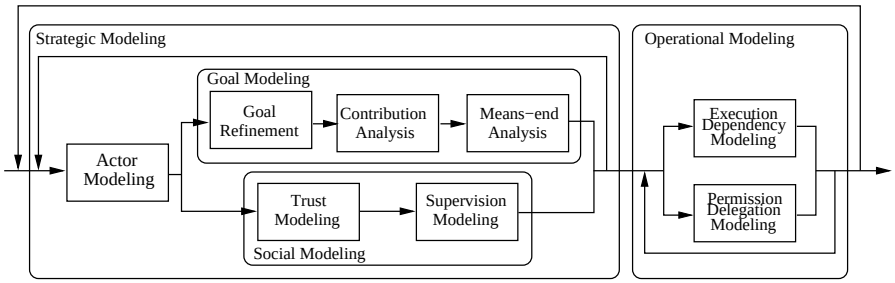


Fig. 10 Requirements Acquisition Process

The diagrams produced during strategic modeling are used as input for operational modeling activities. These activities intend to discover and establish the relationships between agents and roles on the basis of the trust diagram and organization chart. The process may require the introduction of new actors to whom goals are assigned. Thereby, the resulting diagrams are further revised through a new iteration of the actor modeling activity. Operational modeling activities can be (partially) automatized by adopting the framework proposed in [12]. This framework uses an off-the-shelf planner to automatically explore design alternatives (i.e., the potential choices that the designer can adopt for the fulfillment of actor objectives) and finding a satisfactory one. Indeed, different actors can be able to achieve the same objective, or different design alternatives can be adopted to achieve the same high level objective. The process ends when all objectives have been dealt with to the satisfaction of the actors who want them.

Fig. 11 presents a fragment of the Smart Item scenario elicited in the course of the project. This figure refines Fig. 6 by considering the agents and roles involved in the provision of medical services and the relations among them.

Organizational Requirements Model Instantiation The aim of this phase is to capture security bugs that may be introduced by the natural tendency to model requirements using only the concept of role. The analysis of security incidents and frauds [7, 24, 42] has revealed that security breaches are often not apparent in policies specified at the organizational level. They only appear at the instance level, when agents are mapped into roles and delegation paths are concretely rolled out. This requires one to compare the structure of the organization (roles and relations among them) with the concrete instance of the organization (agents playing some roles in the organization and relations among them).

To address this issue, we have defined a procedure that automatically determines the objectives, entitlements and capabilities of agents as well as the relations among them with respect to their position within the organization and the actual responsibilities and permissions assigned to each agent. The procedure takes in input a set of facts, describing the requirements model,[3] and returns a number of answer sets (i.e.,

[3] Intuitively, graphical models are encoded as sets of facts. A description of the mapping is presented in the next section.

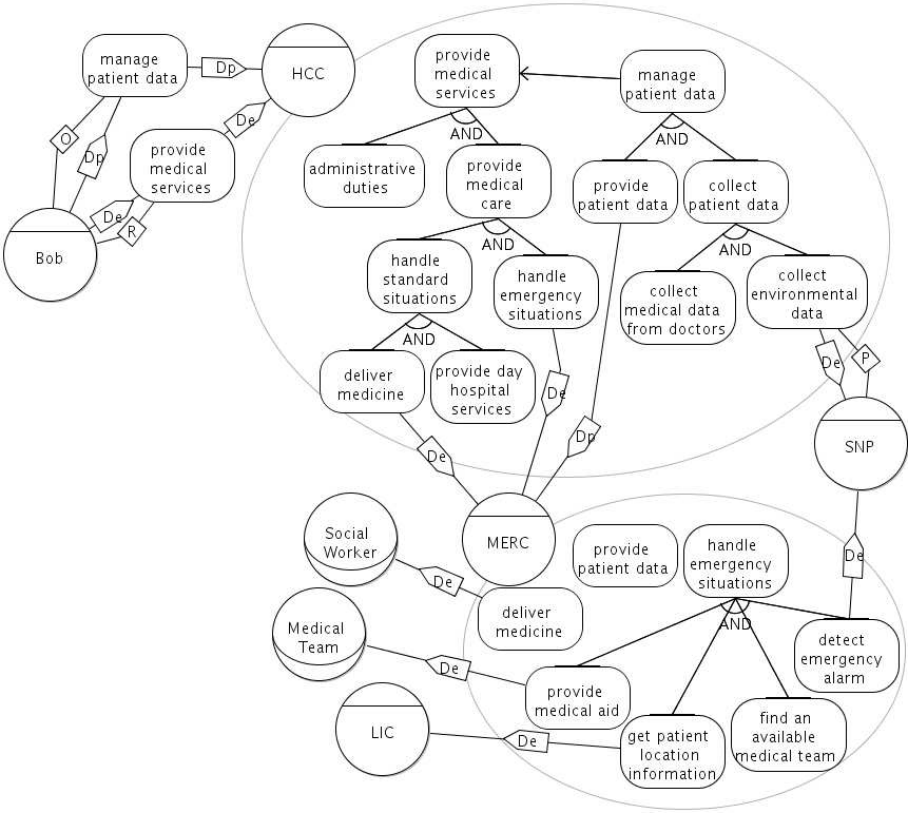


Fig. 11 Smart Item requirements model

sets of facts) that represent all possible configurations (i.e., assignments of responsibilities and permissions to agents) of the system. Here, we present the intuition behind the procedure and refer to [55] for the formal rules that implement it.

Goals are instantiated differently depending on the concepts they are associated to (i.e., objectives, entitlements, and capabilities). An agent is interested in fulfilling a particular instance. Looking at the e-Business scenario, a customer desires to protect the privacy of his financial information, but he only cares about the privacy of his own information. The instantiation procedure creates a new instance of the goal and adds a fact stating that such an instance is an objective of the agent. One may argue that there might be situations in which an agent wants to achieve more than one instance of the goal. This is, for example, the case of the bank which is in charge of providing loans to different clients. These are, however, assigned objectives as the client is the loan requester. Capabilities are instantiated differently. An agent that has the capabilities to achieve a goal, has the capabilities to achieve all instances of the goal. For instance, if Paul is a pre-processing clerk of the bank, he has the capability to identify all clients of the bank rather than only one particular

client. In this case, the instantiation procedure adds a set of facts stating that the agent has the capability to achieve all existing instances of the goal.

The instantiation of entitlements is done on a case-by-case basis. The designer could assign a permission to an agent (or a role) with the intended meaning that the agent (or the agents playing the role) is entitled to control only a particular instance of the goal. For instance, when a designer says that customers own customer personal information, he means that every customer owns only his personal data. However, there may be situations in which an agent owns all instances of a goal. This is the case of the bank director that, as legal representative of the bank, has full authority concerning all executions of the loan process performed within the bank. The intrinsic difference in the use of ownership has demanded for a refinement of the concept to automatize the implementation procedure. In particular, we distinguish between existential ownership and universal ownership. *Existential ownership* models situations in which the agent is the legitimate owner of a particular instance of a goal. For instance, it is used to represent that a customer is the owner of customer personal information. *Universal ownership* models situations in which the agent is the legitimate owner of all the instances of a goal. For instance, it is used to represent that the bank director is the owner of the loan process performed within the bank. Accordingly, the instantiation procedure creates a new instance of the goal and adds a fact stating that the agent is the owner of that instance when the requirements engineers has specified an existential ownership, and adds a set of facts stating that the agent is the owner of all existing instances of the goal when the requirements engineers has specified a universal ownership.

Relations between actors are also instantiated differently depending on their type. Permission delegations and execution dependencies represent the transfer of the entitlements and objectives of actors to other actors. Accordingly, the procedure instantiates such relations on the basis of the instances of the dependum, which the depender wants to achieve, and of the delegatum, which the delegator is entitled to achieve. However, the instantiation procedure treats them differently when they involve roles. When the delegatee is a role, the procedure generates one answer set in which all agents playing that role are authorized to accomplish the delegatum. This choice is based on the intuition behind the RBAC model [45] for which every user assigned to a role inherits the permissions associated with that role. When the dependee is a role, only one agent playing that role is appointed to perform the assigned duties. This intuition is actually closer to reality than one may think: when the bank appoints a pre-processing clerk to identify a certain client, only one clerk is appointed to perform the activities concerning the identification of that client. In this case, the procedure generates a number of answer sets equal to the number of agents playing the role of dependee. Each answer set represents a configuration of the system in which only one agent is appointed to accomplish the dependum.

Finally, trust relations model the expectations that an actor has concerning the fair behavior and dependability of another actor in achieving a goal or delivering a resource. We assume that such expectations are not related to a specific instance of the trustum but refer to the general behavior of the trustee. Thus, the instantiation procedure instances the relation for all existing instances of the trustum. If the trustor

Availability
P1 Actors delegate the execution of their (direct or indirect) objectives only to actors that they trust.
P2 Requesters can satisfy their objectives.
P3 Requesters are confident that their objectives will be satisfied.
Confidentiality
P4 Actors that can access resources have been authorized by the legitimate owners.
Authorization
P5 Actors delegate permissions on their (direct or indirect) entitlements only to actors they trust.
P6 Owners are confident that their entitlements are not misused.
P7 Actors, who delegate permissions to achieve a goal or furnish a resource, have the right to do so.
Availability & Authorization
P8 Requesters can achieve their objectives.
P9 Requesters are confident to achieve their objectives.
P10 Providers have all the permissions necessary to achieve assigned responsibilities.
Privacy
P11 Actors have only the permissions necessary to achieve assigned responsibilities.

Table 1 Properties of Design

(trustee) is a role, the procedure generates a fact, representing an instance of the relation, for each agent playing the role of trustor (trustee).

Requirements Verification Once the requirements model has been captured and instantiated, the purpose of Secure Tropos is to assist system designers in verifying the availability, confidentiality, authorization, and privacy of the designed system and the consistency of security and privacy requirements with functional requirements. To support system designers in this task, we have defined a number of properties (Table 1). P1 verifies if every execution dependency is matched by a chain of trust of execution relations, that is, if there is a path composed of trust of execution relations from the depender to the dependee. P2 verifies if requesters have assigned their objectives to actors that have the capabilities to achieve them. P3 strengthens P2 by verifying that every dependency chain from the requester to providers, who have committed the achievement of requester objectives or part of them, is matched by a trust (of execution) chain. P4 verifies if an actor to whom a resource is delivered, has been authorized by the owner of the resource. P5 is the permission counterpart of P1. P6 extends P5 by verifying that every permission delegation chain rooted in the owner is matched by a trust (of permission) chain. P7 verifies if actors delegate the permission to achieve a goal or to furnish a resource, for which they have been previously authorized. P8 extends P2 by considering the notion of permission: it verifies if a requester has appointed providers that have also the permission to achieve requester (sub)objectives. P9 extends P3 along the same lines. P10 verifies the requirements model from the perspective of providers: it checks if a provider has the permissions necessary to achieve assigned responsibilities. Finally, P11 verifies if permissions are granted to actors who actually need it to perform their duties. If all properties are not simultaneously satisfied, vulnerabilities may occur in the actual implementation of the system. For instance, the assignment of responsibilities to untrusted actors or to actors that do not have the capabilities to achieve them can compromise the availability of the system.

To assist designers during requirements verification, we have developed a formal framework based on Answer Set Programming (ASP) [28]. Graphical models are

transformed into ASP specifications, which are passed to an external solver together with axioms that formally define the semantics of SI* concepts. The mapping consists of representing every entity (i.e., agent, role, goal, and resource) and relation (i.e., request, execution dependency, trust of execution, AND/OR decomposition, etc.) in the model as a fact, defined as a predicate that identifies the entity or relation, together with the appropriate number of well-typed[4] arguments. Axioms are ASP rules and are used to complete the extensional description of the system. Properties of design (Table 1) are encoded as ASP constraints. For the lack of space, we refer to [22] for the complete list of axioms and their description, and to [32] for details on the transformation of graphical models into ASP specifications. Axioms are also used to encode the instantiation procedure presented in Section 3.2.

In essence, ASP solvers produce program completion, compute model of a completion, and verify if the model is indeed an answer set. If a property is not satisfied, the solvers return a warning that includes the information needed to localize the problem in the model. Such information allows requirements engineers to understand the problem and drives them in the selection of an appropriate solution. Inconsistencies might be due to either unspecified requirements or conflicting requirements. Their resolution is a condition necessary for the development of secure systems [51]. Detecting and solving inconsistencies aid requirements engineers to detect implicit and unspecified requirements, understand system vulnerabilities, and identify and evaluate solutions to reduce vulnerabilities. If inconsistencies are identified, system designers can revise the requirements model either by reconstructing the organizational structure of the system or by adopting protection mechanisms. Accordingly, the requirements model is used as input for a new iteration either of the requirements acquisition process or of security mitigation as shown in Fig. 9.

Security Mitigation Security mitigation is intended to ensure that the system is designed to operate at a level of security consistent with the potential risks that can result from the failure in the achievement of objectives or misuse of entitlements. When properties of design are not satisfied by the model, system designers (together with stakeholders) have to revisit requirements (and models) to make sure that the implementation of the system is not affected by vulnerabilities.

To address this issue, we have adopted an approach based on security patterns. Security patterns [48] have been proposed to assist system designers in identifying and formulating security measures that are relevant to the development of secure systems. They provide standardized solutions in systematic and structured manner to deal with recurring security problems. Security patterns aid designers, included not security experts, in identifying and understanding security concerns, and in implementing appropriate security measures. Usually, patterns are described by the context in which the pattern should be implemented, the problem addressed by the pattern, and the solution to the problem.

In this work, we have employed the Security and Dependability (S&D) pattern library [5, 15] developed in the context of the SERENITY project. In this pattern library, the context is specified as a SI* model. The problem is defined in terms of

[4] Well-typed means that arguments have the same type of those requested by the predicate.

Context. The Data Controller outsources the execution of data processing to an outside Supplier for which Data Subject's personal data are needed. However, the Data Subject has signed a contract according to which only the Data Controller and assigned members of its organization are entitled to process his data.
Requirement. The Supplier shall have the permission necessary to achieve outsourced data processing.
Solution. Before the Data Controller can outsource the data processing to the Supplier, he has to obtain the consent of the Data Subject. The consent can be seen as a contract establishing what and how data will be processed by the Supplier. The Data Controller must also ensure, preferably by a written agreement, that the Supplier strictly follows all conditions relating to data processing that were imposed on him.

Table 2 Outsourcing Pattern

the properties of design presented in Table 1. The solution is specified in different forms depending on its level of application; it can be, for example, a new SI* model, a set of constraints, or a workflow. Below we present a security pattern enforcing legal requirements when outsourcing data processing [15].

Example 7. Outsourcing is the transfer of management control of business activities to an outside supplier. This business strategy is adopted by organizations to reduce costs, but has a strong impact on their security and privacy requirements. From a privacy perspective, the data controller must seek the consent of data subjects for disclosing their personal data to third parties and ensure them that the data processor processes those data according to privacy regulations. The pattern addressing these outsourcing requirements is presented in Table 2.

If an inconsistency is spotted during requirements verification, designer can browse the S&D pattern library looking for a pattern to be applied. Once a pattern has been selected, its implementation consists of instantiating the pattern solution and then plugging the instantiated solution in the requirements model.

Example 8. The requirements model of the Smart Item scenario in Fig. 11 shows that the HCC outsources the monitoring of patient health status to the Sensor Network Provider (SNP). The analysis performed during requirements verification has revealed that the SNP cannot achieve assigned responsibilities due to the lack of the necessary permissions (i.e., violation of P10). Among the patterns in the S&D library, the outsourcing pattern addresses this issue where Bob is the Data Subject, the HCC is the Data Controller, and the SNP is the Supplier. Fig. 12 presents the fragment of Fig. 11 to which the pattern solution has been applied.

The next step of the requirements analysis process is to verify the impact of the applied pattern solution on the system. We want assurance that introduced mechanisms effectively fix the vulnerabilities identified. At the same time, we also need to ensure that neither their introduction affects system functionality, nor they introduce new vulnerabilities. To this end, the revised requirements model is checked again through a new iteration of the requirements verification process (Fig. 9).

Example 9. The analysis performed on the revised model of Fig. 12 has shown that the outsourcing pattern solves the problem concerning SNP's lack of the permission. However, the analysis has also revealed that the implementation of the pattern can introduce new security issues. For instance, the HCC may want assurance that the SNP does not repudiate the data processing agreement.

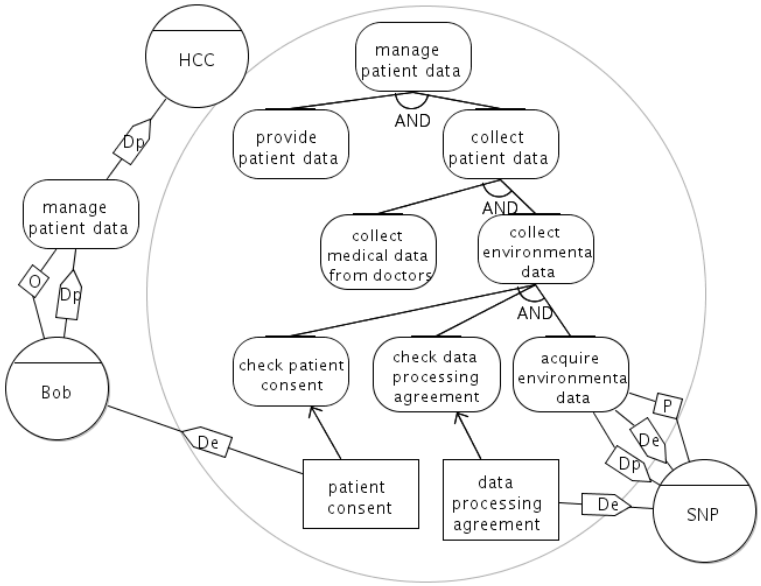


Fig. 12 Requirements model after the application of the pattern

4 Consolidation by Case Studies

The reader familiar with our initial work [21, 22] may find out a number of similarities but also a large number of changes. Addressing the challenges posed by the industry case studies and above all *the challenge of having industry partners themselves using the modeling framework* has driven us to make them as we shall see later. In particular, the interaction with industry partners has demonstrated the importance of defining the proper terminology in order to avoid misunderstanding, besides identifying the concepts necessary to capture issues faced in industry.

The first and maybe foremost change over i* and our initial proposal is the disappearance of the concepts of actor and position from the language. The initial version of SI*, as well i*, supported the concept of generic actor as well its refinements: agent, role, and position. The general feeling when dealing with industry partners (who has never seen a goal oriented model beforehand) was that the less and clearer constructs the better. The interaction with industry partners has revealed that the notion of generic actor is ambiguous. The recurring questions from them were: “What is the distinction between actor and agent? and between role and actor?”, “Should I use actor here or role?”. In contrast, most people understood intuitively well the distinction between agent and role. More important, the notion of generic actor is badly suited for formal modeling. SI*, as well as i*, models involve two different levels of analysis: *organizational* and *instance*. At the organizational level, the structure of organizations is defined associating with each role the objectives, entitlements and capabilities related to its duties within the organization. At the instance level,

the focus is on single agents. They not only are defined along with their objectives, entitlements and capabilities, but they are also defined in terms of the roles they can play. Some case studies have demanded for a clear separation between the organizational and instance level in order to capture conflicting requirements and security issues. For instance, separation of duty properties in the e-Business case study (see also [46]) can be captured only by looking at the distinction between organizational requirements and their instantiation. By definition, the concept of generic actor is well placed neither at the organizational level nor at the instance level. The notion of position was dropped for similar reasons. Actually, it is defined as an aggregation of roles and we have noted that our partners hardly used such a concept. Rather, they largely preferred to introduce a role with component and specialization relations. Such models could be easily done using *is-a* and *is-part-of* relations.

However, hierarchies based only on component and specialization relations do not completely capture the structure of organizations. Schaad and Moffett [47] demonstrate that organizational control is fundamental to prevent and detect errors and frauds in the business processes carried out by organizations. Some early papers on Role-Based Access Control (RBAC) [39, 45] also argue that in some cases the inheritance of permissions downward organizational hierarchies may be undesirable but these seminal observations had no follow-up in later RBAC models. To address these issues, we have introduced the concept of *supervision* proposed by Moffett [39]. Supervision consists of several activities including monitoring, evaluation, advising, and review. Once this construct is introduced, modeling the *organization chart*, a task essentially impossible in i*, becomes extremely easy.

The first contribution of our framework was the introduction of the notions of objective and ownership to distinguish essentially who wants a goal to be fulfilled and who is entitled to decide who can fulfill the goal and how the goal can be fulfilled. The term ownership has raised problems in the understanding of the concept and its difference with the concept of objective. Actually, many industry partners have interpreted the expression "owning a goal" as the desire of an actor to achieve the goal. For this purpose, we have tried to use different wording for specifying such a concept, also discussing with colleagues from the faculty of Law, but at the end always fell back to the terminology of "owning a goal". Case studies in which privacy concerns play a key role (such as the Smart Item case study and the privacy policy described in [34]) called for the introduction of the notion of capability: the ability of an actor to fulfill a goal, even if he does not want it, nor it is authorized to fulfill it. The three notions make it possible to capture concepts such as need-to-know, data subject, data processor, and data controller, which are well grounded in the privacy legislation (e.g., the EU Directive on data protection, 95/46/EC).

The i* modeling framework has been designed with cooperating IT systems in mind. Thus, a dependency between two actors means that the dependee will take responsibility for fulfilling the objective of a depender, and the dependee is implicitly authorized to do so. Both assumptions might not be true as soon as we start modeling security and trust requirements. To this end, we have introduced the notions of permission delegation. The separation between delegation and trust then makes it possible to model situations in which an actor grants the permission to achieve a

goal or furnish a resource to untrusted actors. The modeling and analysis of the case studies have revealed the same concerns for what concerns responsibilities.

Example 10. In the Smart Item scenario, the HCC is responsible for the delivery of medicines to patients and appoints a social worker to do it. According to art. 1228 of the Italian Civil Code ("Liability for acts of auxiliaries"), data controllers who avail third parties in the execution of data processing are also liable for their malicious, fraudulent, or neglect acts, unless otherwise intended by the parties. Unless there is a trust relation between the HCC and the social worker, the health care provider might adopt some measure to verify if the social worker has delivered medicines and take compensation actions in case the job has not been done properly.

Secure Tropos extends Tropos to support the analysis of security requirements. Thereby, the methods, tools, and processes offered by Tropos should also be revised and modified to accomplish these purposes. The introduction of new concepts in the language has necessarily required the introduction of new modeling activities that explain how these concepts contribute to the capture of the requirements model. This is not, however, the only difference. A main change over Tropos and the initial proposal of Secure Tropos is the clear separation between strategic and operational modeling. This choice is driven by the interest of our partners in the definition of the business processes that implement the requirements model. Separating operational aspects of the system allows designers to use the corresponding model as a start point for the implementation of requirements models in terms of business processes.

Another difference is when actor capabilities are identified. In Tropos, capability modeling starts at the end of the architectural design. Though this proposal might be adequate to design software agents, it is not appropriate when modeling and analyzing secure socio-technical systems. First, without this information it is not possible to verify the compliance of the requirements model with availability and need-to-know properties. Moreover, capabilities of human agents cannot be "prescribed" in advance. Differently from software agents, human agents can only be "described".

The main contribution of Secure Tropos was the support for the formal analysis of security, privacy, and trust requirements. In this work, we have introduced two additional phases, namely organizational requirements model instantiation and security mitigation. The aim of this phase is to support industry partners, who are not security expert, in securing the designed system. Organizational requirements model instantiation has been proposed to support a more accurate requirements analysis to address organizational control issues such as separation of duty and conflicts of interest can be only captured at the instance level.

A difficulty we met during the projects was in the requirements elicitation and modeling. Our previous experience (e.g., [34, 36]) revealed that the modeling phase may be very laborious and time-consuming if requirements specifications are completely unstructured. To support and drive industry partners during requirements elicitation and modeling, we have provided them with a *requirements collection schema* [4]. This schema is constructed using existing techniques from requirements engineering [44] and is intended to represent organizational and system requirements in a semi-structured way. The aim of the schema is to bridge the gap between

requirements specified in natural language and the SI* language. The requirements modeling process thus comprises two steps. The first step consists of filling the requirements collection schema from natural language scenario description. For this purpose, we have investigated the application of natural language techniques to automate this step. A result of that study is reported in [27]. Once the requirements collection schema is filled, the modeling process proceeds by producing a SI* model on the basis of the collected information. The drawing of SI* models from the filled schema is straightforward as the template contains a table for each SI* concept.

We have also provide industry partners with a CASE tool [32] during requirements modeling and analysis. The tool provides requirements engineers with a graphical framework for the creation of security requirements models using SI* concepts; support for translating graphical models into formal specifications; and a front-end to state-of-the-art, off-the-shelf ASP solvers for requirements verification.

5 Related Work

Recent years have seen the emergence of standards for capturing security and privacy aspects of IT systems. For instance, XACML [41] defines schemes for the specification of access control policies. Another standard is EPAL [3] which defines a language for formalizing enterprise-internal privacy practices. This language specifies privacy policies expressing whether certain actions are allowed or not, using elements such as data users, actions, purposes, conditions, and obligations. P3P [16] is intended to formalize privacy statements that are published by an enterprise. Its aim is to enable web sites to express their privacy practices in a standard format that can be retrieved automatically and interpreted easily by user. These proposals, however, do not provide facilities for modeling the structure of an organization together with organizational goals. This is critical issue since access control and privacy policies might not be consistent with organizational and system requirements.

On the other hand, there are several proposals that have approached the problem of organization modeling and analysis. For instance, the Enterprise Project [50] attempts to capture an enterprise-wide perspective of organizations and to drive them in making strategic, tactical and operational decisions. The Enterprise Engineering Methodology [11] provides a framework for the development of an enterprise strategy synchronized with organizational goals. The Computer-Integrated Manufacturing Open System Architecture (CIMOSA) [1] integrates enterprise operations by means of efficient information exchange. Enterprises are analyzed from four perspectives: the functional structure required to satisfy the objectives of an enterprise and related control structures; the information required by each function; the resources and their relations to functional and control structures; and the responsibilities assigned to individuals for functional and control structures. The Generalised Enterprise Reference Architecture and Methodology (GERAM) [9] defines a set of basic concepts to describe the structure, content, and behavior of enterprises. Such

concepts enable the modeling of the human component in an enterprise operation as well as the parts of a business process and their supporting technologies.

Among MAS proposals, OperA [18] attempts to design models of organizations that support dynamic and autonomous interactions by focusing on agent societies. This proposal uses the agent paradigm to provide a natural way to view and characterize intelligent organizational systems. To model different roles, goals, and interactions within an organization, the framework adopts a three-layer approach: the organizational model describes the intended behavior and overall structure of the society from the perspective of the organization in terms of roles, interactions, and social norms; the social model instantiates the organizational model with specific agents mapped into roles through a social contract; finally, the interaction model describes society agent interactions by means of interaction contracts. MOISE+ [25] focuses on the structure and functionality of organizations, and the deontic relation between them to explain how a MAS achieves its purpose. Accordingly, the organizational specification is formed by a structural specification, a functional specification, and a deontic specification. The structural specification adopts the concepts of role, role relation, and group to model the individual, social, and collective structural levels of organizations. The functional specification is based on the concepts of mission and global plan. The deontic specification then links the structural specification to functional specification in terms of permissions and obligations.

We notice a gap between the proposals coming from security engineering and organization engineering. From one side, we have complex methodologies for analyzing organizations and their IT systems, but their focus is mostly on system functionality rather than on security. On the other side, we have security technologies, but they do not offer any methodological support for design decision making. Several efforts were spent to close the gap between security requirements and security technologies by addressing security concerns during the system development process. For instance, Jürjens [26] proposes UMLsec, a UML profile designed to model security related features such as confidentiality and access control within UML diagrams. Its objectives are to encapsulate knowledge and make it available to developers in the form of a widely used design notation, and to provide formal techniques for security requirements verification. Basin et al. [8] propose SecureUML, an UML-based modeling language for specifying access control policies and integrating them into a model-driven software development process. Similar approaches have been proposed by Doan et al. [19], who incorporate Mandatory Access Control (MAC) into UML, and by Ray et al. [43], who model RBAC as a pattern using UML diagram template. McDermott and Fox [38] adapt use cases to capture and analyze security requirements, and call these abuse cases. An abuse case is an interaction between a system and one or more actors, where the results of the interaction are harmful to the system or to one of the stakeholders of the system. Sindre and Opdahl [49] defined misuse cases, the converse of UML use cases, which describe uses that the system should not allow. One of the major limitations of all these proposals is that they are intended to model only a computer system and the policies and security mechanisms it supports. Modeling system protection mechanisms alone is not sufficient. Conflicts and loopholes at the interface between the organization and

its IT systems are a major source of vulnerabilities. Moreover, specifying security requirements as security mechanisms and architectural constraints may constrain architectural decisions too early, resulting in inappropriate security solutions.

Moving towards early requirements, Zave [56] treats security as a vague goal to be satisfied, while a precise description and enumeration of specific security properties and behavior is missing. Van Lamswerde extends KAOS [17] to address security issues by introducing the notions of obstacle to capture exceptional behaviors [52] and anti-goals to model intentional obstacles set up by attackers to affect security goals [51]. Anti-goals are defined as the negation of security goals such as confidentiality, availability, and privacy and represent the goals of attackers. Anti-goals are refined to form an attack tree on the basis of attackers capabilities as well as software vulnerabilities. Security requirements are defined as the countermeasures to software vulnerabilities or anti-requirements, that is, anti-goals that are realizable by some attacker. Along the same line, Liu et al. [31] refine the i* modeling framework [54] by analyzing attackers, dependency vulnerabilities among actors and possible countermeasures. All actors are assumed to be potential attackers who inherit capabilities and intentions from the corresponding legitimate actor. Dependency analysis is used to identify the vulnerable points in the dependency network. During countermeasure analysis, designers investigate how to protect the system from attackers and vulnerabilities. Elahi et al. [20] extend this work by focusing on how attackers can compromise the system by exploiting vulnerabilities that software components and organizational procedures bring to the system. These proposals are complementary to our work as they tackle security issues from a different prospective: they identify and analyze possible attackers and their goals, and how those attackers can compromise the system, whereas our work mainly focuses on authorization, trust, and privacy requirements within an organization. For instance, in [35] we show how Secure Tropos can assist policy writers in the specification and analysis of access control policies on the basis of organizational and system requirements. Another difference lies in the approach for security analysis. For instance, in [20, 31, 56] the authors model security requirements as "non-functional" or "quality" requirements and intend to identify the security solutions that contribute to their satisfaction. On the contrary, we have introduced concepts specific to security and propose to verify if requirements models satisfy security properties of design.

6 Conclusions

In this chapter, we have presented a modeling framework and a security requirements engineering methodology for modeling and analyzing security aspects of IT systems and their organizational context. They are presented by focusing on how the application to case studies has driven their definition.

The research presented here is still in progress. Much remains to be done to further refine the language and methodology to support the full fledged industrial usage. We are currently extending the language to capture behavioral aspects of the

system for the architectural and detailed design phases. This extension has two implications. On one hand, it makes it possible to capture more sophisticated security properties. On the other hand, such concepts support the (semi-)automatic derivation of business processes (in the sense of BPEL code fragments) from the requirements.

Another direction under investigation involves the enrichment of SI* with concepts necessary for capturing more sophisticated privacy concerns. Several researchers have recently proposed frameworks for specifying and enforcing privacy policies [3, 16, 41]. They provide the right concepts to capture privacy policies, but they do not support policy writers in the analysis of organizational and system requirements. Our objective is to bridge the gap between requirements analysis and policy specification by deriving privacy policies from requirements.

System designers usually are neither security nor legal experts. Thereby, they may have difficulties in deploying systems that comply with security and privacy requirements as defined in the current legislation. We are enhancing the S&D pattern library to better support them. We are also defining a pattern integration schema to drive designers in the application of safe combinations of patterns where potential interferences and conflicts between patterns may occur.

Acknowledgements This work has been partially funded by EU SENSORIA and SERENITY projects, and by the MIUR FIRB TOCAI project.

References

1. AMICE Consortium: Open System Architecture for CIM. Springer-Verlag (1993)
2. Anderson, R.: Security Engineering: A Guide to Building Dependable Distributed Systems. Wiley (2001)
3. Ashley, P., Hada, S., Karjoth, G., Powers, C., Schunter, M.: Enterprise Privacy Authorization Language (EPAL 1.1). Research Report 3485, IBM Research (2003). URL http://www.zurich.ibm.com/security/enterprise-privacy/epal
4. 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)
5. Asnar, Y., Bonato, R., Giorgini, P., Massacci, F., Meduri, V., Riccucci, C., Saidane, A.: Secure and Dependable Patterns in Organizations: An Empirical Approach. In: Proc. of RE'07. IEEE Press (2007)
6. Asnar, Y., Giorgini, P., Massacci, F., Zannone, N.: From Trust to Dependability through Risk Analysis. In: Proc. of ARES'07, pp. 19–26. IEEE Press (2007)
7. Association of Certified Fraud Examiners: The 2006 report to the nation (2006)
8. Basin, D., Doser, J., Lodderstedt, T.: Model Driven Security: from UML Models to Access Control Infrastructures. TOSEM **15**(1), 39–91 (2006)
9. Bernus, P., Nemes, L.: A Framework to Define a Generic Enterprise Reference Architecture and Methodology. Computer Integrated Manufacturing Systems **9**(3), 179–191 (1996)
10. Bresciani, P., Giorgini, P., Giunchiglia, F., Mylopoulos, J., Perini, A.: TROPOS: An Agent-Oriented Software Development Methodology. JAAMAS **8**(3), 203–236 (2004)
11. Bryce, M., Associates: PRIDE-EEM Enterprise Engineering Methodology (2006). Available at http://www.phmainstreet.com/mba/pride/eemeth.htm

12. Bryl, V., Massacci, F., Mylopoulos, J., Zannone, N.: Designing Security Requirements Models through Planning. In: Proc. of CAiSE'06, *LNCS*, vol. 4001, pp. 33–47. Springer-Verlag (2006)
13. Castelfranchi, C., Falcone, R.: Principles of trust for MAS: Cognitive anatomy, social importance and quantification. In: Proc. of ICMAS'98, pp. 72–79. IEEE Press (1998)
14. Chung, L.K., Nixon, B.A., Yu, E.S.K., Mylopoulos, J.: Non-Functional Requirements in Software Engineering. Kluwer Publishing (2000)
15. Compagna, L., El Khoury, P., Massacci, F., Thomas, R., Zannone, N.: How to capture, communicate, model, and verify the knowledge of legal, security, and privacy experts: a pattern-based approach. In: Proc. of ICAIL'07, pp. 149–154. ACM Press (2007)
16. Cranor, L., Langheinrich, M., Marchiori, M., Reagle, J.: The Platform for Privacy Preferences 1.0 (P3P1.0) Specification. W3C Recommendation (2002). URL http://www.w3.org/TR/P3P/
17. Dardenne, A., van Lamsweerde, A., Fickas, S.: Goal-directed Requirements Acquisition. Sci. of Comp. Prog. **20**, 3–50 (1993)
18. Dignum, V.: A model for organizational interaction: based on agents, founded in logic. Ph.D. thesis, Universiteit Utrecht (2004)
19. Doan, T., Demurjian, S., Ting, T.C., Ketterl, A.: MAC and UML for secure software design. In: Proc. of FMSE'04, pp. 75–85. ACM Press (2004)
20. Elahi, G., Yu, E.: A goal oriented approach for modeling and analyzing security trade-offs. In: Proc. of ER'07, LNCS 4801, pp. 375–390. Springer (2007)
21. Giorgini, P., Massacci, F., Mylopoulos, J., Zannone, N.: Requirements Engineering for Trust Management: Model, Methodology, and Reasoning. Int. J. of Inform. Sec. **5**(4), 257–274 (2006)
22. Giorgini, P., Massacci, F., Zannone, N.: Security and Trust Requirements Engineering. In: FOSAD 2004/2005, LNCS 3655, pp. 237–272. Springer-Verlag (2005)
23. Guarda, P., Massacci, F., Zannone, N.: E-Government and On-line Services: Security and Legal Patterns. In: Proc. of MeTTeg'07 (2007)
24. House of Lords: Prince Jefri Bolkiah vs KPMG. 1 All ER 517 (1999)
25. Hübner, J.F., Sichman, J.S., Boissier, O.: A Model for the Structural, Functional, and Deontic Specification of Organizations in Multiagent Systems. In: Proc. of SBIA'02, pp. 118–128. Springer-Verlag (2002)
26. Jürjens, J.: Secure Systems Development with UML. Springer-Verlag (2004)
27. Kiyavitskaya, N., Zannone, N.: Requirements Model Generation to Support Requirements Elicitation: The Secure Tropos Experience. ASE (2008)
28. Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S., Scarcello, F.: The DLV System for Knowledge Representation and Reasoning. TOCL **7**(3), 499–562 (2006)
29. Li, N., Grosof, B.N., Feigenbaum, J.: Delegation logic: A logic-based approach to distributed authorization. TISSEC **6**(1), 128–171 (2003)
30. Liu, L., Yu, E., Mylopoulos, J.: Analyzing Security Requirements as Relationships Among Strategic Actors. In: Proc. of SREIS'02 (2002)
31. 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 (2003)
32. Massacci, F., Mylopoulos, J., Zannone, N.: Computer-Aided Support for Secure Tropos. ASE **14**(3), 341–364 (2007)
33. Massacci, F., Mylopoulos, J., Zannone, N.: An Ontology for Secure Socio-Technical Systems. In: Handbook of Ontologies for Business Interaction, chap. XI. The IDEA Group (2008)
34. Massacci, F., Prest, M., Zannone, N.: Using a Security Requirements Engineering Methodology in Practice: The compliance with the Italian Data Protection Legislation. CSI **27**(5), 445–455 (2005)
35. Massacci, F., Zannone, N.: A Model-Driven Approach for the Specification and Analysis of Access Control Policies. In: Proc. of IS'08, *LNCS*, vol. 5332, pp. 1087–1103. Springer-Verlag (2008)
36. Massacci, F., Zannone, N.: Detecting Conflicts between Functional and Security Requirements with Secure Tropos: John Rusnak and the Allied Irish Bank. In: Social Modeling for Requirements Engineering. MIT Press (2008). To appear

37. Mayer, R.C., Davis, J.H., Schoorman, F.D.: An integrative model of organizational trust. Acad. Management Rev **20**(3), 709–734 (1995)
38. McDermott, J., Fox, C.: Using Abuse Case Models for Security Requirements Analysis. In: Proc. of ACSAC'99, pp. 55–66. IEEE Press (1999)
39. Moffett, J.D.: Control principles and role hierarchies. In: Proc. of RBAC'98, pp. 63–69. ACM Press (1998)
40. Mouratidis, H., Giorgini, P., Manson, G.: Integrating security and systems engineering: Towards the modelling of secure information systems. In: Proc. of CAiSE'03, vol. 2681, pp. 63 – 78. Springer-Verlag (2003)
41. OASIS: eXtensible Access Control Markup Language (XACML) Version 2.0. OASIS Standard (2005)
42. 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)
43. Ray, I., Li, N., France, R., Kim, D.K.: Using UML to visualize role-based access control constraints. In: Proc. of SACMAT'04, pp. 115–124. ACM Press (2004)
44. Robertson, S., Robertson, J.: Mastering the requirements process. ACM Press/Addison-Wesley Publishing Co. (1999)
45. Sandhu, R.S., Coyne, E.J., Feinstein, H.L., Youman, C.E.: Role-Based Access Control Models. IEEE Comp. **29**(2), 38–47 (1996)
46. Schaad, A., Lotz, V., Sohr, K.: A model-checking approach to analysing organisational controls in a loan origination process. In: Proc. of SACMAT'06, pp. 139–149. ACM Press (2006)
47. Schaad, A., Moffett, J.: Separation, review and supervision controls in the context of a credit application process: a case study of organisational control principles. In: Proc. of SAC'04, pp. 1380–1384. ACM Press (2004)
48. Schumacher, M., Fernandez, E.B., Hybertson, D., Buschmann, F., Sommerlad, P.: Security Patterns - Integrating Security and Systems Engineering. John Wiley & Sons (2005)
49. Sindre, G., Opdahl, A.L.: Eliciting security requirements with misuse cases. REJ **10**(1), 34–44 (2005)
50. Stader, J.: Results of the Enterprise Project. In: Proc. of BSC SGES'96 (1996)
51. van Lamsweerde, A.: Elaborating security requirements by construction of intentional anti-models. In: Proc. of ICSE'04, pp. 148–157. IEEE Press (2004)
52. van Lamsweerde, A., Letier, E.: Handling Obstacles in Goal-Oriented Requirements Engineering. TSE **26**(10), 978–1005 (2000)
53. Yu, E., Cysneiros, L.: Designing for Privacy and Other Competing Requirements. In: Proc. of SREIS'02 (2002)
54. Yu, E.S.K.: Modelling strategic relationships for process reengineering. Ph.D. thesis, University of Toronto (1995)
55. Zannone, N.: A Requirements Engineering Methodology for Trust, Security, and Privacy. Ph.D. thesis, University of Trento (2007)
56. Zave, P.: Classification of research efforts in requirements engineering. CSUR **29**(4), 315–321 (1997)