International Journal of Software Engineering and Knowledge Engineering
© World Scientific Publishing Company

The SI* Modeling Framework: Metamodel and Applications

Nicola Zannone
Department of Information and Communication Technology, University of Trento, Italy
zannone@dit.unitn.it

Received (Day Month Year)
Revised (Day Month Year)
Accepted (Day Month Year)

Security Requirements Engineering is emerging spurred by the realization that security must be dealt from the early phases of the system development process. Modeling languages in this field are challenging as they must provide concepts appropriate in order to talk about security within an organization.

In previous work we introduced the SI* modeling language tailored to capture security aspects of socio-technical systems. SI* is founded on four main notions, namely supervision, permission, delegation, and trust. In this paper, we present the SI* metamodel. We also present some frameworks and methodologies founded on this modeling language for the analysis of security and dependability requirements as well as the exploration of design alternatives and the generation of skeletons of secure business processes. The paper also presents a development environment that uses the SI* metamodel as its basis core.

1. Introduction

The last years have seen the need of capturing security aspects of information systems from the early phases of the system development process. Some proposals [8, 25] adopted traditional Requirements Engineering (RE) frameworks as they are to capture security facets of IT systems. However, these proposals lack fundamental concepts specific to security. Thereby, they cannot be used to capture a number of situations that were frequent in industrial case studies but cumbersome if not impossible to express with existing constructs. Other proposals [5, 22, 31, 34] enhanced off-the-shelf RE frameworks with constructs tailored to capture protection aspects of security. One of the major limitations of these proposals is that they deal with security services (integrity, availability etc.) and related protection mechanisms (such as passwords, or cryptographic mechanisms). In contrast, to understand security issues we need to model the organization and social relationships between all actors involved in the system.

It is generally accepted in the RE research community that system development requires models that represent the system-to-be along with its operational environment. This is even more important when the system has to meet security requirements, since security breaches often occur at an organizational level rather than at a technical one [1].

To allow for a systematic design of secure socio-technical systems, we have developed the SI* modeling language [26]. This language is intended to capture security facets of the system-to-be and its organizational environment. SI* is based on the i*/Tropos modeling framework [6, 38], where specifications employ basic primitives such as actor, goal, task, resource and social dependency between two actors, and enhances such a framework with four main notions, namely permission, delegation, trust, and supervision, which are at the very foundation of all security concerns. The SI* modeling framework has been tested in several provincial, national, and European projects. One of these projects is the EU project SERENITY[a] where SI* was used to analyze domains of interest for the project's purposes, ranging from Health Care and Air Traffic Management (ATM) domains to e-Government and e-Business domains, in collaboration with industrial partners. The analysis of such domains intends to support security engineers in the definition of organizational security solutions. Another project is the FIRB TOCAI.IT project,[b] where SI* was applied to identify, model, and analyze novel enterprise organizational models of integration, coordination, cooperation and interoperability and their possible enhancement related to the integration of IT technologies in the organizational process. In the course of these projects, the language has been applied to a number of industrial case studies [3, 9, 21, 27, 28]. These applications of the language have shown that the identified primitives are adequate to capture most of the security issues we encountered. Indeed, the proposed constructs were up the challenge and revealed a number of pitfalls, especially when formal analysis techniques were applied.

Some work [6] presented the metamodel for the i*/Tropos modeling framework. However, such a metamodel cannot be used as it is to support SI* because of the new proposed concepts and some changes of existing concepts demanded by the industrial case studies to which SI* has been applied. The objective of this paper is to define the SI* metamodel, pointing out the differences with the original i*/Tropos metamodel as well as giving the motivations for changes. We also show how this modeling language has been adopted by some frameworks and methodologies for the analysis of security and dependability analysis as well as the exploration of design alternatives and the generation of the skeleton of secure business processes. Finally, we present a development environment that uses the SI* metamodel as its basis core. To make the discussion more concrete, the SI* language is applied to model an ATM scenario, focusing on the coordination of activities among different actors collaborating to manage the airspace. This ATM scenario is taken from one of the case studies analyzed within the EU SERENITY project.

[a]System Engineering for Security & Dependability – 27587 IST-FP6-IP-SERENITY – http://www.serenity-project.org
[b]Knowledge-oriented technologies for enterprise aggregation in Internet – RBNE05BFRK FIRB-TOCAI.IT – http://www.dis.uniroma1.it/˜tocai/

The paper is organized as follows. Section 2 informally describes the concepts offered by the SI* modeling language along with the models in which such concepts are used. Section 3 presents its metamodel. Section 4 gives a brief overview of the frameworks and methodologies that have adopted SI*. Section 5 presents a CASE tool developed to draw SI* models. Finally, Section 6 discusses related work and Section 7 concludes the paper and discusses possible directions for future work.

2. Concepts and Models

The SI* modeling framework [26] has been proposed for the design of secure socio-technical systems. It extends and refines the i* modeling language [38] with basic primitives suitable for capturing security aspects of organizations. i* employs basic primitives such as "actor", "goal", "task", "resource", and "social dependencies between actors". SI* extends i* by introducing primitives for modeling *entitlements* of actors and making explicit their *capabilities*. Moreover, the language adopts the notion of *delegation* to model the transfer of rights between actors, and the notion of *trust* to model the expectation of an actor about the behavior of other actors. Together with concepts, SI* proposes a number of modeling activities to capture the design of secure socio-technical systems; each of them produces a diagram that represents a view of the requirements model.

An *actor diagram* describes domain stakeholders and system's actors. Actors can be modeled using two types of sub-units: *agent*, which is an actor with concrete, physical manifestations and can be used to refer to human as well as software agents and organizations, and *role*, which is the abstract characterization of the behavior of a social actor within some specialized context. An agent is said to *play* a role. Using roles and agents alone is not enough to capture many important features demanded by the industrial case studies to which SI* has been applied, and, in particular, the rich organizational structure. To this extent, SI* (as the ancestor i*) employs constructs for modeling actor hierarchies based on the concepts of *component* and *specialization*. For instance, the concept of component can be used to identify the members of an organization as well as the sub-components of a software agent. The concept of *specialization* can be used to model role hierarchies. A role is a specialization of another if it refers to more specialized activities. In this setting, all specialized sub-roles inherit all properties of the generalized super-role. Actors are described in terms of their objectives, entitlements, and capabilities. *Objectives* are goals intended to be achieved, tasks intended to be executed, or resources required by the actor. *Entitlements* are goals, tasks and resources that the actor has the permission to achieve, execute, and furnish respectively. *Capabilities* are goals, tasks, and resources that the actor is able to respectively achieve, execute, and furnish.

A *social diagram* describes the social relations among stakeholders and system's actors in terms of the organizational structure and expectations about the behavior of other actors. To this intent, the SI* modeling framework uses the concepts of *supervision*, *trust of permission*, and *trust of execution* together with the concepts of agent and role presented above. The basic idea underlying supervision is that, if a role supervises another role, the first is responsible for the behavior of the latter. This concept is used to build organizational hierarchies. Trust of execution is used to model the expectations of one actor has on the competence and dependability of another actor. In general, by trusting in execution, an actor is sure that the other actor will achieve the goal, perform the task, or furnish the resource. Trust of permission is used to model the expectations of one actor that another actor does not misuse a goal, a task, or a resource. By trusting in permission, an actor is sure that the (possibly) granted permission is properly used.

An *execution dependency diagram* represents the execution dependency network among actors. An *execution dependency* indicates that one actor appoints another actor to achieve a goal, execute a task, or furnish a resource. This would be matched, for instance, by a call to an external procedure. In this setting, the dependee will take care of the achievement of the goal, execution of the task, or supply of the resource. At the same time, the depender becomes vulnerable. Indeed, the depender has no warranty that the dependee will achieve the goal, execute the task, or furnish the resource.

A *permission delegation diagram* represents the permission delegation network among actors. A *permission delegation* indicates that one actor authorizes another actor to achieve a goal, execute a task, or use a resource. This would be matched by the issue of a delegation certificate, such as digital credential or a letter. In this setting, the delegatee is entitled to achieve the goal, execute the task, or use the resource. At the same time, the delegator becomes vulnerable. Actually, the delegator has no warranty that the delegatee will not misuse the granted permission.

A *goal/task diagram* represents the analysis of the rationale of single actors and appears as a balloon attached to a specific actor. Goal analysis is performed using three techniques, namely *goal/task decomposition*, *contribution analysis*, and *means-end analysis*. Goal/task decomposition analyzes and refines root goals and tasks to build a finer goal/task structure in terms of AND and OR decompositions. Contribution analysis studies the impact of the achievement of goals and execution of tasks on the system. Means-end analysis aims at identifying the tasks that can be used to achieve a goal as well as the resources produced and consumed by a task.

3. Metamodel

Metamodels play a key role in Model Driven Engineering [4]. Actually, metamodels assist system designers in managing the complexity of developing complex, enterprise-level systems. A metamodel is essentially "a model of a modelling language" [13], that is, a collection of concepts and their relationships. Metamodels essentially provide an explicit description (constructs and rules) of how a domain-specific model is built. They aid system designers to verify if their models are conforming to the modeling language and to define a formal semantics as well as to ease the definition of possible future extensions of the language.

The need of a metamodel was also recognized in the Tropos community. Bresciani et al. [6] have proposed a metamodel of Tropos methodology, an agent-oriented software engineering methodology that adopts i* as modeling framework. Such a metamodel defines the basic concepts of actor, goal, task, resource, and social dependency. Tropos, as many other traditional goal oriented methodologies such as KAOS [11], have a clear cut notion of goals: a strategic interest of an actor that the design should possibly fulfill. Here we must broaden the notion of goal because system designers must be able to model situations where actors who have the capabilities to achieve a goal are different from the one who have the permission to do it and both are different from the actors who want the goal fulfilled. As consequences, the metamodel should be extended and revised in order to capture this intuition. In the remainder of this section, the metamodel of the SI* modeling language is defined in terms of UML class diagrams.

The concepts related to the actor diagram are presented in Fig. 1. The basic idea behind the i* was the concept of actor, namely an active entity that has strategic goals and performs actions to achieve them. Though we use the term "actor" informally throughout the paper, there is no place for it in SI* since it does not allow for a clear distinction between the analysis of organizational structures and their instantiations [17], making it difficult to detect security issues such as conflicts of interest [18]. Instead, two different constructs have been adopted in the language: *role* and *agent*. Such concepts are represented as UML classes. An agent can *play* $0..n$ roles and a role can be played by $0..n$ agents. The *is-part-of* relation define a part/whole relationship between two agents (or roles), meaning that an agent (role respectively) is a component with structure of another agent (role respectively). A role can be specialized in $0..n$ roles through relation *is-a*. *Own*, *request*, and *provide* are binary relations represented as UML classes. *Own* relates an *owner* (i.e., an agent or a role) and an *entitlement* (i.e., a goal, a task, or a resource). We assume that a goal, a task, or a resource can be owned by at most one actor. *Request* relates a *requester* (i.e., an agent or a role) and an *objective* (i.e., a goal, a task, or a resource). *Provide* relates a *provider* (i.e., an agent or a role) and a *capability* (i.e., a goal, a task, or a resource). Here different actors can request or provide the same goal, task, or resource.

Fig. 2 presents a fragment of the actor diagram concerning the ATM scenario studied in the SERENITY project. An ATM system is managed by an Air Traffic Control Center (ACC), which provides air traffic control services in a particular airspace. The airspace is divided in sectors, each of them is managed by a Sector Team. Each Sector Team is compound of an Executive Controller (EC) and a Planning Controller (PC). An EC is assigned to manage the traffic in the sector and has the capability to provide pilots with safe instructions. A PC is assigned to manage the traffic incoming to the sector. In our scenario, the airspace was divided in three sectors and, consequently, we have three Sector Teams together with the corresponding ECs and PCs. In particular, the role ACC 1-A EC is played by Edison and the role ACC 1-A PC is played by Paul.

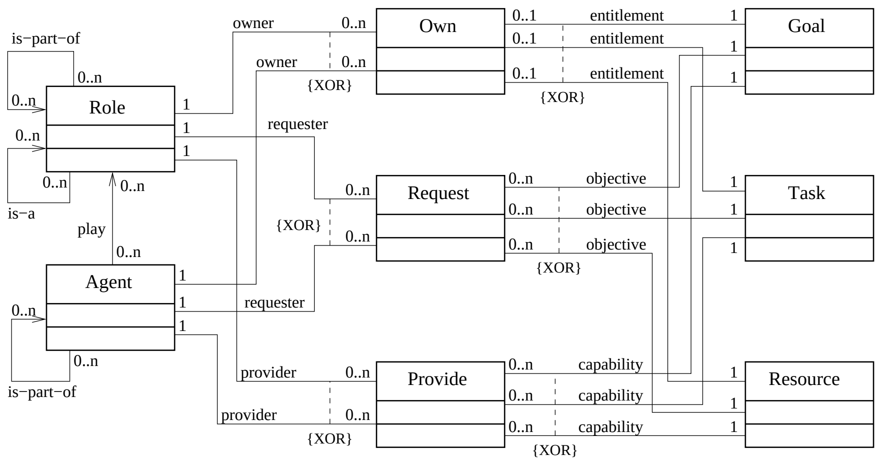


Fig. 1. Metamodel for Actor Diagram

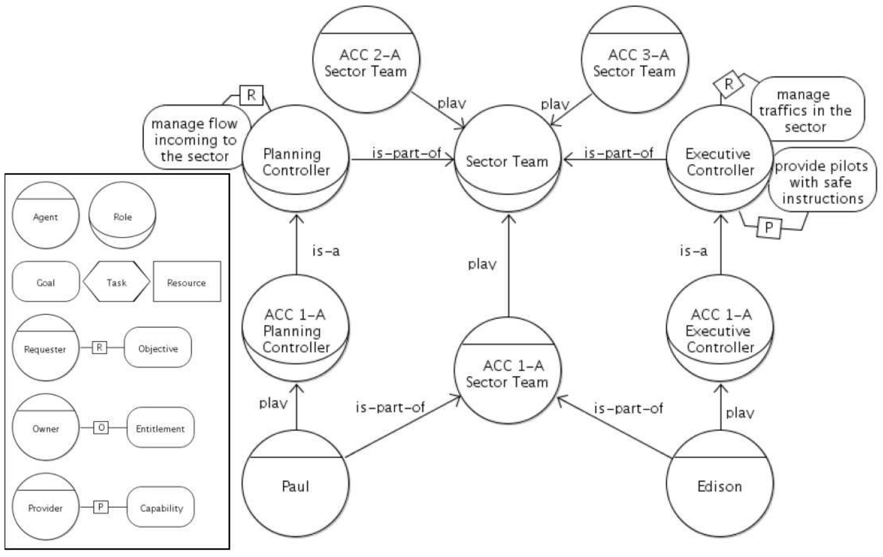


Fig. 2. Actor Diagram for the ATM scenario

Fig. 3 shows the fragment of the metamodel concerning the social diagram. A *supervisor* (i.e., a role) can supervise the behavior of 0..n subordinates and a *subordinate* (i.e., a role) can be supervised by 0..n supervisor. Trust relations are ternary relationships and relate a *trustor* (i.e., an agent or a role), a *trustee* (i.e., an agent or a role), and a *trustum* (i.e., a goal, a task, or a resource). They are also associated with two attributes – *condition* and *depth* – that are used to control the validity of trust relations and the building of trust chains. A condition is a boolean expression that determines the validity of the trust relation. Depth represents how much an actor takes into account the judgment of another actor. Depth can be either a positive integer or "*" for unbounded depth. Depth equal to 1 means that an actor trusts another actor, but he does not trust the trustee's judgment; depth equal to 2 means that an actor trusts another actor, and all actors the trustee trusts directly, and so on. Finally, unbounded trust means that an actor trusts another actor unconditionally. For the sake of compactness, we do not report in Fig. 3 the part of the metamodel concerning agents because it is similar to the one for role. The only difference is the lack of the *supervise* relation between agents.

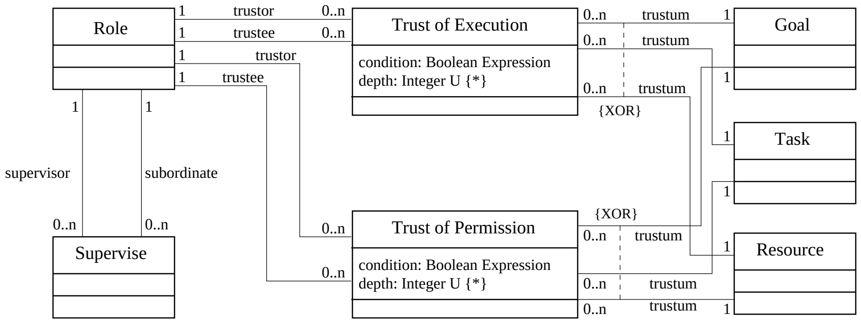


Fig. 3. Metamodel for Social Diagram

Fig. 4 presents a fragment of the social diagram concerning the ATM scenario. Here the Supervisor is appointed to supervise the behavior of both the EC and PC. Moreover, the EC trusts that the PC will define a traffic plan.

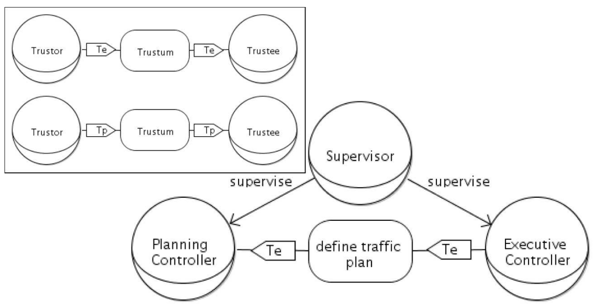


Fig. 4. Social Diagram for the ATM scenario

Fig. 5 presents the metamodel underlying execution dependency and permission delegation diagrams. An *execution dependency*, also called *execution delegation*, is a ternary relation and relates a *depender* (i.e., a role or agent), a *dependee* (i.e., a role or and agent), and a *dependum* (i.e., a goal, a task, or a resource). A *permission* *delegation* is a ternary relation and relates a *delegator* (i.e., a role or agent), a *delegatee* (i.e., a role or and agent), and a *delegatum* (i.e., a goal, a task, or a resource). As trust relationships, execution dependencies and permission delegations have attributes *condition* and *depth.* Conditions determine the validity of relationships. Depth can be seen as the number of re-delegation steps that are allowed; depth 1 means that no re-delegation is allowed, depth n means that $n-1$ further step are allowed, and depth "*" means that unbounded re-delegation is allowed.

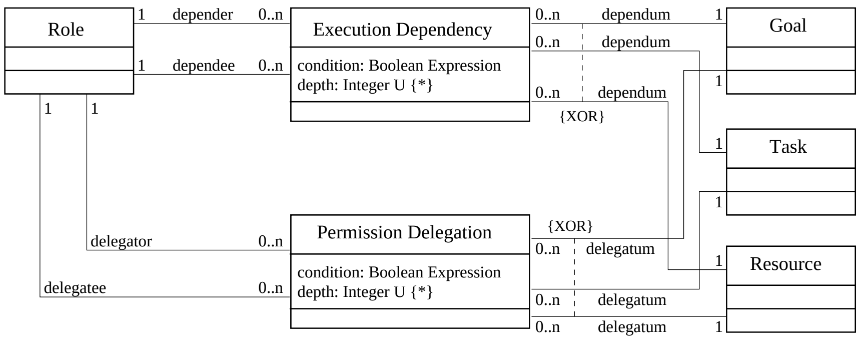


Fig. 5. Metamodel for Execution Dependency and Permission Delegation Diagrams

Fig. 6 shows a fragment of the execution dependency and permission delegation diagrams for the ATM scenario. The ACC A Supervisor appoints the ACC 1-A PC to manage traffic in sector 1-A and the ACC 1-A EC to control traffic in sector 1-A. To achieve assigned duties, the ACC 1-A PC needs a planning traffic display, for which he depends on the ACC A Supervisor. The ACC A Supervisor authorizes the ACC 1-A PC to use it. The ACC 1-A EC needs a traffic plan and depends on the ACC 1-A PC for its definition. Finally, when the traffic incoming in the sector increases over a certain threshold, the ACC A Supervisor can delegate the control of part of the airspace to another supervisor, who is requested to define a partial delegation schema.

Fig. 7 presents the metamodel defining goal/task diagrams. The central concepts of goal, task, and resource are represented by classes *Goal*, *Task*, and *Resource*, respectively. These concepts can be analyzed from the *point of view* of an actor using *means-end analysis*, *AND/OR decomposition*, and *contribution analysis.* Means-end analysis is a ternary relationship that relates an actor (i.e., a role or an agent), whose perspective is analyzed, an *end* (i.e., a goal), and a *means* (i.e., a task). This relationship is also used to represent the resources *produced* and *consumed* by a task. *AND-* and *OR-decompositions* are ternary relationships that refine a *root* goal/task into *subgoals/subtasks* with respect to the perspective of one actor. Notice that the SI* language, differently from i*, allows one to decompose a goal only into subgoals and a task only into subtasks, forbidding, for instance the decomposition of tasks in

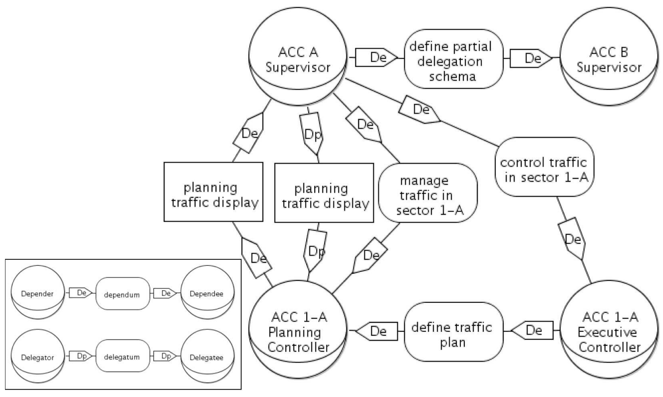


Fig. 6. Execution Dependency and Permission Delegation Diagrams for the ATM scenario

subgoals. This choice is due to the need of clearly separating the strategic model (i.e., goals) from the operational model (i.e., tasks and resources). Contribution analysis is a ternary relationship that relates an actor, whose perspective is analyzed, and two goals/tasks. This relationship is associated with an attribute – *type* – that defines the impact that a goal/task has on the other goal/task. Such an impact can be full positive (denoted by ++), partial positive (denoted by +), full negative (denoted by −−), or partial negative (denoted by −).

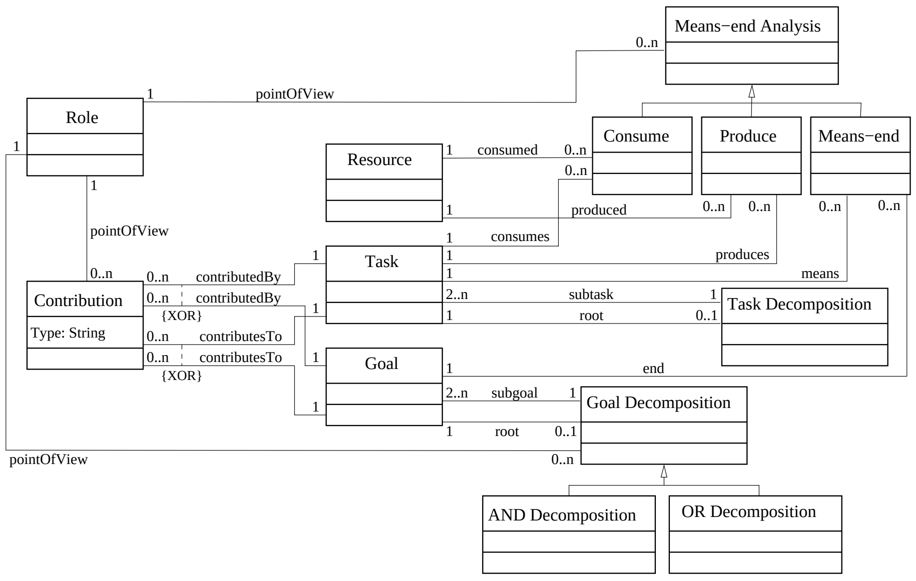


Fig. 7. Metamodel for Goal/Task Diagram

Fig. 8 analyzes goal manage traffic incoming to the sector from the perspective of the PC. This goal is AND-decomposed into two subgoals: ensure traffic to be conflict free and maintain traffic at an acceptable level. To achieve the former, the PC can adopt task define traffic plan, which is AND-decomposed into subtasks monitor traffic, plan traffic, and coordinate changes of flight plans. To execute task plan traffic, the PC needs a planning traffic display. To achieve goal maintain traffic at an acceptable level, the PC can adopt task manage traffic forecast, which is AND-decomposed into subtasks monitor traffic forecast and response traffic forecast. The PC needs a forecast display for the execution of the former task.

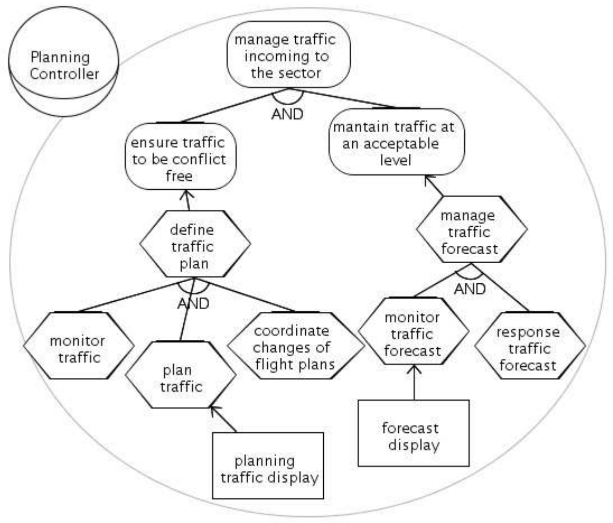

Planning Controller
manage traffic incoming to the sector
AND
ensure traffic to be conflict free
mantain traffic at an acceptable level
define traffic plan
manage traffic forecast
AND
AND
monitor traffic
plan traffic
coordinate changes of flight plans
monitor traffic forecast
response traffic forecast
forecast display
planning traffic display

Fig. 8. Goal Diagram for the ATM scenario

4. Applications

The SI* modeling language was originally defined to support the Secure Tropos methodology [19] in capturing and modeling security and functional requirements of socio-technical systems. Secure Tropos aims to assist system designers in ensuring that the system is designed to operate at an adequate level of security. In particular, the methodology provides support for the verification of the compliance of requirements models with a number of security properties and the consistency between security and functional requirements as well as for driving designers in the adoption of appropriate security measures.

In particular, the verification phase is based on a number of properties that have been defined with the intent of verifying the availability, confidentiality, authorization, and privacy of the designed system [19]. These properties are defined as combinations of the concepts presented in the previous section. For instance, availability properties verify if a requester (possibly indirectly) depends on actors who have the capabilities to accomplish the assigned duty. Confidentiality properties verify if actors, who can access a resource, has been authorized by the owner of that resource to use it. The methodology also provides a formal framework based on Answer Set Programming (ASP) [23] for a tool-supported verification [19]. Specifically, graphical models are encoded into ASP programs, which are passed to external ASP engines together with the axioms that formally define the semantics of SI* concepts and a set of constraints that encode security properties. ASP engines produce program's completion, compute model of a completion, and verify if the model is indeed an answer. The failure of properties will drive the designer in the adoption of appropriate measures for securing the system.

Successively, the SI* modeling language has been adopted and adapted by other frameworks and methodologies to perform different kinds of analysis. For instance, Bryl et al. [7] have used a planning approach to automatically explore design alternatives (i.e., the potential choices that the designer may adopt for the fulfillment of system actors' objectives) and finding a satisfactory one. Actually, an actor may not have the capabilities to fully achieve his objectives by itself, so he can either appoint some actors to fulfill them entirely or decompose them, and assign part of them to other actors. Different actors can be able to achieve the same objectives, or different alternatives can be adopted to achieve the same objective. Assignments of responsibilities and the consequent permissions are often driven by the expectations an actor has about the behavior of other actors, where the depender (the delegator respectively) will usually prefer to assign the achievement of its objectives (grant permission on its entitlements respectively) to actors he trusts. In this setting, a SI* model is conceived as networks of execution dependencies and permission delegations among actors. Intuitively, these can be seen as *actions* that the designer has ascribed to the members of the socio-technical system. As suggested by Gans et al. [16], the task of designing such networks can then be framed as a planning problem for multi-agent systems: selecting a suitable possible design corresponds to selecting a plan that satisfies the prescribed or described goals of human or system actors. Accordingly, the generation of design alternatives can be accomplished by a planner which is given as input a set of actors, their objectives, and the trust relations among them, and generates alternative multi-agent plans to fulfill all given goals. The work in [7] essentially proposes to use off-the-shelf planners to select among the potential dependencies and delegations the actual ones that will constitute the final choice of the requirements engineer.

Asnar et al. [2, 3] proposed the Tropos Goal-Risk (GR) framework, a framework for modeling, assessing, and treating risks on the basis of the likelihood and severity of failures. The GR framework originally adopted and adapted the Tropos Goal Model [20] to catch the idea of the three layers analysis introduced by Feather et al. [14] in their Defect Detection and Prevention (DDP) framework. Accordingly, the GR framework consists of three conceptual layers: goal, event, and treatment. The *goal layer* analyzes the goals of each actor and identifies which tasks the actor needs to perform to achieve the goals and corresponds to the goal/task diagram presented in previous sections. The *event layer* models uncertain events along their effects to the goal layer. The *treatment layer* identifies specific tasks (also called treatments) that should be introduced to mitigate the effect of the event layer on the goal layer. These three layers allow one to reason about uncertain events that obstruct goals and evaluate the effectiveness of treatments. In [3], the GR framework has been refined for assessing risks of socio-technical systems, borrowing the concepts of execution dependency and trust of execution from SI*. The basic idea is that assignments of responsibilities are typically driven by the level of trust towards other actors [12, 33]. Accordingly, a low level of trust increases the risk perceived by the depender about the achievement of his objectives. Using this framework, an actor can thus assess the risk in delegating the achievement of his objectives and decide whether or not the risk is acceptable.

The GR framework uses concepts and relationships (e.g., events, impact relations, alleviation relations, etc.) that are not present in SI* so that this language and, consequently, the metamodel have to be enhanced to accommodate such concepts. Fig. 9 shows a fragment of the new part of the SI* metamodel concerning risk analysis. *Events* are uncertain circumstances (out of the control of actors) that can have an *impact* on the fulfillment of goals or execution of tasks. As in Fault Tree Analysis [35], an event can be analyzed using decomposition relations. The *severity* of events is represented as the sign of *impact* relations, similarly to contribution analysis. The GR framework distinguishes the tasks used to achieve goals (*functional task*) or to treat the effects of events (*treatment*). Treatments can be used either to *mitigate* the likelihood of events or to *alleviate* the impact that events have on goals and tasks. Tasks have attribute *cost*, which defines how many efforts (e.g., in terms of money) are needed to execute the task. Events, tasks and goals are also characterized by two attributes: SAT and DEN. Such attributes represent the values of evidence that the goal can be satisfied and denied respectively. Their values are qualitatively represented in the range of {*(F)ull,(P)artial,(N)one*}, with the intended meaning $F > P > N$. Finally, we want to point out that events can be local, i.e., they are perceived by single actors, or global, i.e., all actors have the same perception of the event. Thereby, the multiplicity of *pointOfView* is different from the one in the metamodel for goal/task diagram.

Fig. 10 shows an example of GR model. Event loss of traffic statistical data is a global event that negatively affects goal define sector configuration. To reduce the likelihood of this event, the Supervisor can adopt treatment install Fault Tolerant system. Event error in Airspace Modeling application is a local event that negatively affects goal define partial delegation schema. The Supervisor can have a default Airspace Model to alleviate the impact of this event.

Finally, we mention the work by Frankova et al. [15]. In this work the authors have used the SI* modeling language as basis for the definition of Secure BPEL, a specification language that extends WS-BPEL [30] for modeling secure business

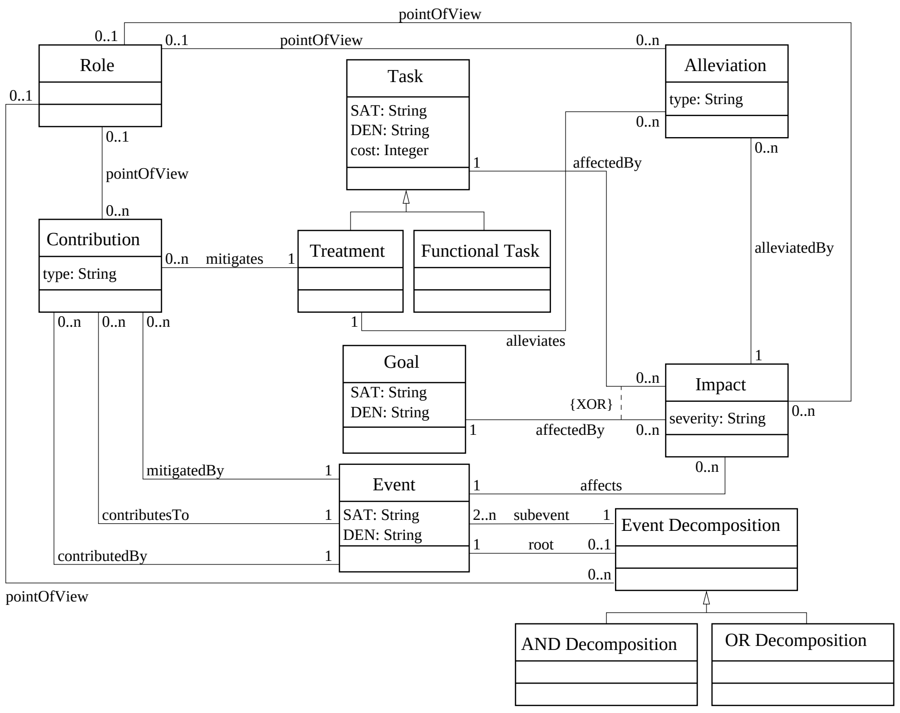


Fig. 9. Metamodel for Goal-Risk Model

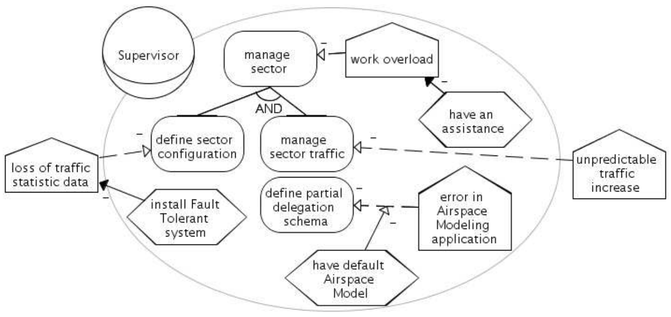

Supervisor
manage sector
work overload
have an assistance
AND
define sector configuration
manage sector traffic
unpredictable traffic increase
loss of traffic statistic data
install Fault Tolerant system
define partial delegation schema
error in Airspace Modeling application
have default Airspace Model
-

Fig. 10. Goal-Risk Model for the ATM scenario

processes. Together with Secure BPEL, the authors have also proposed a refinement methodology to assist business analysts in the derivation of the skeleton of secure business processes (expressed in Secure BPEL) from requirements models (expressed in SI*). Their work was motivated by the need of bridging the gap between requirements engineering methodologies and the actual development of secure software and secure business processes based on the Service-Oriented Architecture.

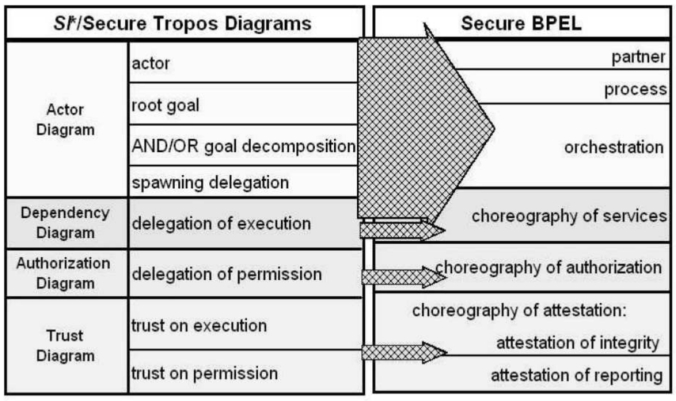


Fig. 11. SI* to Secure BPEL

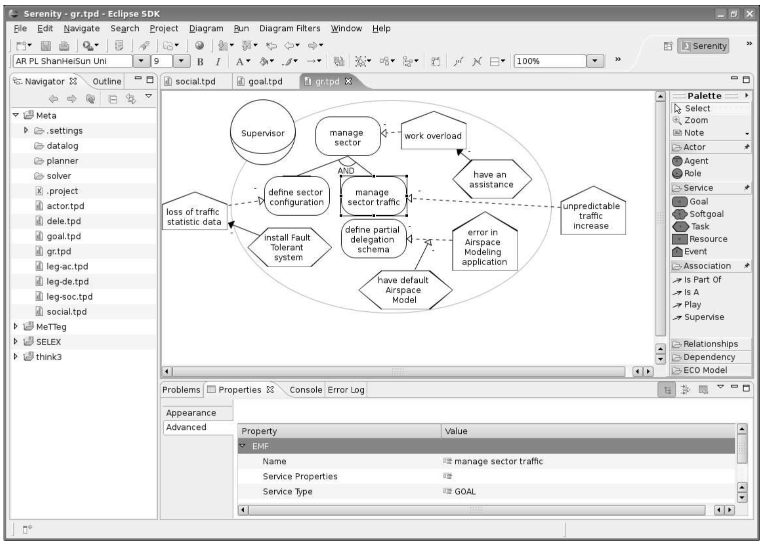
To this intent, the authors have defined a mapping to refine SI* concepts into Secure BPEL specifications (Fig. 11 [15]) together with guidelines to drive business analysts during the refinement process. This approach thus aims to ensure that business processes meet the business goals that have motivated their definition and the security requirements that have to be implemented by them.

5. A Modeling and Analysis Tool

A CASE tool supporting the SI* modeling language has been developed in the course of the SERENITY project.[c] This tool is an Eclipse plugin designed to model and verify security and dependability requirements of socio-technical system. Specifically, the tool provides a modeling framework that can be used to draw SI* models (enriched with GR concepts) as well as front-end to external solvers for the verification of designed models.

The modeling framework takes advantage of the use of off-the-shelves Eclipse plugins, namely the Graphical Editing Framework[d] (GEF), which allows developers to create a rich graphical editor from existing application models, the Eclipse Modeling Framework[e] (EMF), which provides developers with modeling framework and code generation facilities for building tools based on a structured data model, and the Graphical Modeling Framework[f] (GMF), which provides a generative component and a runtime infrastructure for developing graphical editors based on EMF and GEF. Fig. 12 shows a screenshot of the Graphical User Interface of the developed plugin: the diagram editor windows on the right, the project and model browser on the left, and the entity properties at the bottom.

[c]A prototype is available at http://sesa.dit.unitn.it/sistar_tool/.
[d]http://www.eclipse.org/gef
[e]http://www.eclipse.org/emf
[f]http://www.eclipse.org/gmf



Property	Value
EMF	
Name	manage sector traffic
Service Properties	
Service Type	GOAL

Fig. 12. Graphical User Interface

Together with a modeling framework, the plugin also supports designers in the verification of security and dependability requirements of socio-technical system. In particular, it allows designers to verify security requirements using the facilities offered by the Secure Tropos methodology and dependability requirements using the facilities offered by the GR framework. For security verification, graphical models are transformed in ASP specifications that are passed to the DVL system [23]. This ASP engine analyzes such specifications to ensure that desirable security properties are satisfied. Violations of properties are visually represented as red edges connecting the entities that have generated the inconsistency. For dependability verification, graphical models are encoded into satisfiability SAT formulas that are verified using the solution proposed in [32].

Finally, the tool offers system analysts with facilities for the generation of graphical models representing the (sub-)optimal design of a system from the description of the system in a tabular format. In particular, the tool provides a number of tables, one for each SI* concept, where the analyst can insert the description of the system along with all possible design alternatives. This description is passed to a planner that, according to what proposed in [7], determines a (sub-)optimal plan listing the actions (execution dependencies, permission delegations, AND/OR decompositions, etc.) that guarantee the achievement of actor's objectives. Based on the identified plan, the tool generates the corresponding graphical SI* model. This model can be successively revised and modified using the Graphical User Interface.

6. Related Work

Security Requirements Engineering is emerging at the crossover between Software Engineering and Security Engineering, spurred by the realization that security must be dealt with during the early phases of the system development process. Proposals for Security Requirements Engineering can roughly be classified under two classes: object-level modeling and meta-level modeling.

Object-level modeling approaches use an off-the-shelves RE framework and model in that framework a number of security requirements. The analysis features of the framework are then used to reason about security or to derive some guidance for the implementation. For instance, the Non-Functional Requirements (NFR) framework [8] treats security requirements as non-functional (or quality) requirements and models them as softgoals, that is, goals having no clear-cut definition. Liu et al. [25] have extended this approach by offering facilities for threats, vulnerabilities and countermeasures analysis. The advantage of this approach is that reasoning about security is virtually cost-free from the perspective of the designer since no new language should be learned and all features of the modeling framework are immediately usable. However, in this approach security notions are indistinguishable from other objects. Therefore, the link between security and functional requirements must be introduced by the designer in ad-hoc manner.

The *meta-level modeling* takes an off-the-shelves RE framework as well as the object-level approach, but enhances it with linguistic constructs to capture security requirements. The metamodel and analysis features of the framework must then be revised to accommodate such constructs. The addition of suitable constructs usually makes the language more intuitive to use. This main advantage is coupled by the possibility of designing analysis features tailored to the security domain.

The need of languages and conceptual models specific to security has brought up a number of proposals especially in UML community. Jürjens proposed UMLsec [22] to model security related features, such as confidentiality and access control. Basin et al. [5] proposed SecureUML, a UML-based modeling language, to integrate information relevant to access control into application models defined in UML. Their approach is focused on modeling RBAC policies and integrating them into a model-driven software development process. A similar approach was proposed by Ray et al. [31]. They proposed to model RBAC policies as a pattern using UML diagram template and represent constraints on RBAC model through the Object Constraint Language. One of the limitations of such proposals is that they are targeted to model a computer system and the policies and access control mechanisms it supports.

Other approaches proposed to model the behavior of attackers. McDermott and Fox adapted use cases to capture and analyze security requirements, and they call the adaption an abuse case model [29]. 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 one of the stakeholders of the system. Along the same lines, Sindre and Opdahl [34] defined misuse cases, the inverse of UML use cases, which describe functions that the system should not allow. Moving towards early requirements, an extension of the KAOS framework is presented in [37] where the notion of obstacle is introduced. KAOS uses the notion of goal as a set of desired behaviors. Likewise, an obstacle defines a set of undesirable behaviors. The negation of obstacles is thus used to determine preconditions for the goal to be achieved. Although obstacles are sufficient for modeling accidental, non-intentional obstacles to security goals, they appear too limited for modeling and resolving malicious, intentional obstacles. To this end, van Lamsweerde [36] introduced the notion of anti-goal to represent intentional obstacles to security goals. Similarly, Crook et al. [10] introduced the notion of anti-requirement to represent the requirements of malicious attackers. The idea is that an anti-requirement is satisfied when the security threats imposed by the attacker are realized. Lin et al. [24] incorporate anti-requirements into abuse frames. The purpose of this framework is to represent security threats and to facilitate the analysis of the conditions in which a security violation occurs.

7. Conclusions

This paper has presented the SI* metamodel that aims to assist requirements engineers in verifying the syntactic consistency of SI* models as well as in defining a formal semantics used for the analysis of security and dependability requirements. We have also shown how the SI* language has been adopted by some frameworks and methodologies for the exploration of design alternatives and the generation of executable business processes. We want to remark that we have extended the metamodel presented in [6] modularly. Therefore, dropping all new proposed features makes one able to draw i*/Tropos models, except for some existing concepts that have been revised during the application of SI* to industrial case studies.

The language presented in this work can be further extended following several directions. An important research direction is to capture behavioral aspects of socio-technical systems. This work is twofold. First, it will allow designers to capture more sophisticated security and dependability properties. Secondly, it can support requirements engineers and business analysts by providing them with tools for the (semi-)automatic derivation of business processes from requirements models.

A second stream of research could investigate concepts tailored to capture privacy facets of socio-technical systems. Several languages (e.g., EPAL and XACML) have been proposed for specifying and enforcing privacy policies. Such languages employ concepts appropriate for modeling privacy policies, such as the ones of purpose and obligation. However, they do not support policy writers in the analysis of organizational requirements and leave them to define privacy policies manually. This work will allow to bridge the gap between requirements analysis and policy specification by providing a means for deriving privacy policies from requirements models.

Acknowledgments

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

References

[1] R. Anderson. Why cryptosystems fail. *CACM*, 37(11):32–40, 1994.

[2] Y. Asnar and P. Giorgini. Modelling risk and identifying countermeasures in organizations. In *Proc. of CRITIS'06*, 2006.

[3] Y. Asnar, P. Giorgini, F. Massacci, and N. Zannone. From Trust to Dependability through Risk Analysis. In *Proc. of ARES'07*, 2007.

[4] U. Aßmann, S. Zschaler, and G. Wagner. Ontologies, Meta-models, and the Model-Driven Paradigm. In *Ontologies for Software Engineering and Software Technology*, pages 249–273. Springer-Verlag, 2006.

[5] D. Basin, J. Doser, and T. Lodderstedt. Model Driven Security: from UML Models to Access Control Infrastructures. *TOSEM*, 15(1):39–91, 2006.

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

[7] V. Bryl, F. Massacci, J. Mylopoulos, and N. Zannone. Designing Security Requirements Models through Planning. In *Proc. of CAiSE'06*, volume 4001 of *LNCS*, pages 33–47. Springer-Verlag, 2006.

[8] L. K. Chung, B. A. Nixon, E. S. K. Yu, and J. Mylopoulos. *Non-Functional Requirements in Software Engineering.* Kluwer Publishing, 2000.

[9] L. Compagna, P. E. Khoury, F. Massacci, R. Thomas, and N. Zannone. How to capture, communicate, model, and verify the knowledge of legal, security, and privacy experts: a pattern-based approach. In *Proc. of ICAIL'07*, pages 149–154. ACM Press, 2007.

[10] R. Crook, D. Ince, L. Lin, and B. Nuseibeh. Security Requirements Engineering: When Anti-requirements Hit the Fan. In *Proc. of RE'02*, pages 203–205. IEEE Press, 2002.

[11] A. Dardenne, A. van Lamsweerde, and S. Fickas. Goal-directed Requirements Acquisition. *Sci. of Comp. Prog.*, 20:3–50, 1993.

[12] R. Falcone and C. Castelfranchi. Social Trust: A Cognitive Approach. In *Trust and Deception in Virtual Societibes.* Kluwer Academic Publishers, 2001.

[13] J.-M. Favre. Foundations of Meta-Pyramids: Languages vs. Metamodels – Episode II: Story of Thotus the Baboon. In *Language Engineering for Model-Driven Software Development*, number 04101 in Dagstuhl Seminar Proceedings. IBFI, Schloss Dagstuhl, Germany, 2005.

[14] M. S. Feather, S. L. Cornford, K. A. Hicks, and K. R. Johnson. Applications of tool support for risk-informed requirements reasoning. *Comput. Syst. Sci. Eng.*, 20(1), 2005.

[15] G. Frankova, F. Massacci, and M. Seguran. From Early Requirements Analysis towards Secure Workflows. In *Proc. of IFIPTM'07*, 2007. The full version appears as Technical Report, DIT-07-036, at http://eprints.biblio.unitn.it/archive/00001220/.

[16] G. Gans, M. Jarke, S. Kethers, and G. Lakemeyer. Modeling the Impact of Trust and Distrust in Agent Networks. In *Proc. of AOIS'01*, pages 45–58, 2001.

[17] P. Giorgini, F. Massacci, J. Mylopoulos, and N. Zannone. Modelling Social and Individual Trust in Requirements Engineering Methodologies. In *Proc. of iTrust'05*,

volume 3477 of *LNCS*, pages 161–176. Springer-Verlag, 2005.

[18] P. Giorgini, F. Massacci, J. Mylopoulos, and N. Zannone. Detecting Conflicts of Interest. In *Proc. of RE'06*, pages 315–318. IEEE Press, 2006.

[19] P. Giorgini, F. Massacci, and N. Zannone. Security and Trust Requirements Engineering. In *FOSAD 2004/2005*, volume 3655 of *LNCS*, pages 237–272. Springer-Verlag, 2005.

[20] P. Giorgini, J. Mylopoulos, and R. Sebastiani. Goal-Oriented Requirements Analysis and Reasoning in the Tropos Methodology. *Eng. Application of Artif. Intell. J.*, 18(2), 2005.

[21] P. Guarda, F. Massacci, and N. Zannone. E-Government and On-line Services: Security and Legal Patterns. In *Proc. of MeTTeg'07*, 2007.

[22] J. Jürjens. *Secure Systems Development with UML*. Springer-Verlag, 2004.

[23] N. Leone, G. Pfeifer, W. Faber, T. Eiter, G. Gottlob, S. Perri, and F. Scarcello. The DLV System for Knowledge Representation and Reasoning. *TOCL*, 7(3):499–562, 2006.

[24] L. Lin, B. Nuseibeh, D. C. Ince, and M. Jackson. Using Abuse Frames to Bound the Scope of Security Problems. In *Proc. of RE'04*, pages 354–355. IEEE Press, 2004.

[25] L. Liu, E. S. K. Yu, and J. Mylopoulos. Security and Privacy Requirements Analysis within a Social Setting. In *Proc. of RE'03*, pages 151–161. IEEE Press, 2003.

[26] F. Massacci, J. Mylopoulos, and N. Zannone. An Ontology for Secure Socio-Technical Systems. In *Handbook of Ontologies for Business Interaction*. The IDEA Group, 2007.

[27] F. Massacci, M. Prest, and N. Zannone. Using a Security Requirements Engineering Methodology in Practice: The compliance with the Italian Data Protection Legislation. *CSI*, 27(5):445–455, 2005.

[28] F. Massacci and N. Zannone. 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, 2006.

[29] J. McDermott and C. Fox. Using Abuse Case Models for Security Requirements Analysis. In *Proc. of ACSAC'99*, pages 55–66. IEEE Press, 1999.

[30] OASIS. Web Services Business Process Execution Language Version 2.0. Public Review Draft, 2006.

[31] I. Ray, N. Li, R. France, and D.-K. Kim. Using UML to visualize role-based access control constraints. In *Proc. of SACMAT'04*, pages 115–124. ACM Press, 2004.

[32] R. Sebastiani, P. Giorgini, and J. Mylopoulos. Simple and minimum-cost satisfiability for goal models. In *Proc. of CAiSE'04*, volume 3084 of *LNCS*, pages 20–35. Springer-Verlag, 2004.

[33] D. Shapiro and R. Shachter. User-Agent Value Alignment. In *Proc. of The 18th Nat. Conf. on Artif. Intell.* AAAI, 2002.

[34] G. Sindre and A. L. Opdahl. Eliciting security requirements with misuse cases. *REJ*, 10(1):34–44, 2005.

[35] M. Stamatelatos, W. Vesely, J. Dugan, J. Fragola, J. Minarick, and J. Railsback. *Fault Tree Handbook with Aerospace Applications*. NASA, 2002.

[36] A. van Lamsweerde. Elaborating security requirements by construction of intentional anti-models. In *Proc. of ICSE'04*, pages 148–157. IEEE Press, 2004.

[37] A. van Lamsweerde and E. Letier. Handling Obstacles in Goal-Oriented Requirements Engineering. *TSE*, 26(10):978–1005, 2000.

[38] E. S. K. Yu. *Modelling strategic relationships for process reengineering*. PhD thesis, University of Toronto, 1995.