An Ontology for Secure Socio-Technical Systems

Fabio Massacci
Dep. of Information and Communication Technology
University of Trento

via Sommarive 14, 38050 Povo (TRENTO), Italy
massacci@dit.unitn.it

John Mylopoulos
Dep. of Information and Communication Technology
University of Trento

via Sommarive 14, 38050 Povo (TRENTO), Italy
jm@dit.unitn.it

Nicola Zannone
Dep. of Information and Communication Technology
University of Trento

via Sommarive 14, 38050 Povo (TRENTO), Italy
zannone@dit.unitn.it

An Ontology for Secure Socio-Technical Systems[i]

ABSTRACT

Security is often compromised by exploiting vulnerabilities in the interface between the organization and the information systems that support it. This reveals the necessity of modeling and analyzing information systems together with the organizational setting where they will operate. In this chapter we address this problem by presenting a modeling language tailored to analyze the problem of security at an organizational level. This language proposes a set of concepts founded on the notions of permission, delegation, and trust. The chapter also presents a semantics for these concepts, based on Datalog. A case study from the bank domain is employed to illustrate the proposed language.

INTRODUCTION

The last years have seen the emergence of standards for capturing security and privacy aspects of information systems (Ashley et al., 2003; Cranor et al., 2002; OASIS, 2005). Those standards provide language constructs but offer no methodological tool for actually making design decisions. In this setting, the inclusion of security features within the system design is usually done after the functional design phase. This is a critical issue since security services and related protection mechanisms have to be fitted into an existing design that might be not able to accommodate them.

It is generally accepted in the Requirements Engineering (RE) research community that system development requires models that represent the system-to-be along with its intended 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 a technical one (Anderson, 1994). Even though there are mature methodologies for modeling and analyzing enterprises and their organizational structure, their focus is mostly on process and marketing aspects, rather than security (AMICE Consortium, 1993; Bryce and Associates, 2006; Dignum, 2004; Yu, 1996; Hübner et al., 2002; Stader, 1996).

Socio-technical system analysis has been proposed to overcome this issue (Emery et al., 1960). This approach aims at capturing the interactions between people and technology in workplaces. In this setting, security is the ability of the system to protect itself against deliberate misbehavior by actors of the organizations involved in the application scenario while still providing expected services when requested by benign actors. For instance, an actor may abuse his position within the organization to gain personal advantages (House of Lords, 1999; Michaely et al., 1999). Therefore, modeling and analyzing the organizational environment where the system will act is crucial for building secure systems. This allows designers to identify security mechanisms that can best protect the system, and their impacts on the system.

This chapter aims at analyzing the problem of modeling security at an organizational level. Based on such an analysis, we identify and formally define basic ontological primitives for modeling organizational and security concepts, paying particular attention to the security relevant social interaction within organizations.

To allow for a systematic design of security in organization, we have developed an agent-oriented requirements engineering methodology, Secure Tropos (Giorgini et al., 2005; Giorgini et al., 2006), tailored to describe both the organizational environment of a system and the system itself. The methodology provides a requirements analysis process that drives system designers from the acquisition of the requirements model up to its verification and validation. One of its main features is the prominent role given to early requirements analysis phase that precedes a prescriptive requirements specification. The main advantage in having such a phase is that one can capture not only the "what" or the "how", but also the "why" a software system is developed. Secure Tropos was originally based on the i* modeling framework (Yu, 1996). This framework has already been used to model and analyze security requirements (Liu et al., 2003). In this work, security requirements are treated as non-functional requirements. This approach supports the representation of design decisions that can contribute to a security goal and the modeling of attackers (both internal and external) who prevent the fulfillment of goals.

However, our work revealed early on that the i* ontology needs to be extended in order to adequately model security, because it lacks fundamental concepts needed in order to talk about security within an organization (Giorgini et al., 2006). To this end, we have proposed an enhanced ontology with three main notions, namely *ownership*, *delegation* and *trust*, which together form the very foundation of all security concerns (Giorgini et al., 2005). Ownership is used to identify goals, tasks and resources that an actor controls; delegation is used to model the transfer of entitlements and responsibilities between actors; finally, trust represents the belief of actors about the behavior of other actors (Mayer et al,. 1995; Rousseau et al., 1998). Once basic ontological primitives have been identified, we develop a comprehensive ontology tailored to model security at an organizational level. To this end, we provide an axiomatic characterization of their intended semantics using Answer Set Programming (Leone et al., 2006). The proposed ontology is intended to serve as the basis for security-related domain ontologies. From an IT perspective, it can serve as a basis for specifying together functional and security requirements.

The chapter is organized as follows. The next section reviews the current state-of-the-art in ontologies for organization and security modeling by presenting the issues in current proposals. Section 3 introduces a bank scenario used as a running example to illustrate the application of the proposed ontology. Section 4 introduces a set of primitive concepts for modeling security at organizational level. Section 5 presents an axiomatic theory of the identified primitives. Section 6 shows how the introduced concepts are enough to detect security vulnerabilities. Finally, Section 7 concludes the paper with some directions for future work.

RELATED WORK

Several research communities have approached the problem of enterprise modeling and analysis, and some of these have addressed issues of security. We discuss below some of the more prominent approaches.

Enterprise Engineering. Organizational modeling of enterprises is often dealt with by enterprise engineering methodologies (AMICE Consortium, 1993; Bernus and Nemes, 1996; Bryce and Associates, 2006; Stader, 1996). Each methodology includes an ontology for modeling organizations, usually supported by a modeling environment and various analysis tools.

Multi-Agent Systems (MAS). Efforts towards modeling organizations have also originated in the MAS community (Dignum, 2004; Hübner et al., 2002). These approaches propose to model multi-agent systems as organizational structures.

Semantic Web. Ontologies constitute basic infrastructure for the Semantic Web. The idea underling Semantic Web proposals is to use shared vocabularies for describing entities of the domain and their inter-relationships (Masolo et al., 2004).

Security Engineering. One of main challenges of security is data protection. Resources must be protected against unauthorized access and/or tampering. This has spurred many researchers to define languages tailored to model privacy and access control policies (OASIS, 2005; Ashley et al., 2003; Cranor et al., 2002).

Enterprise engineering approaches tackle the issues of organizational analysis and modeling from an enterprise perspective. For instance, the Enterprise Project (Stader, 1996) aims to capture an enterprise-wide perspective of organizations. Such models are intended to drive enterprises in making strategic, tactical and operational decisions. To achieve a high degree of integration, the Enterprise Project proposed the Enterprise Ontology (Uschold et al., 1998) which includes a set of terms often used to describe enterprises. This ontology focuses on organizational structure, strategy, activities and processes, as well as marketing aspects. The Enterprise Engineering Methodology (Bryce and Associates, 2006) provides a framework that allows the study of an organization and the development of an enterprise strategy synchronized with organizational goals. The methodology includes an ontology for specifying priorities within an organization, along with plans for implementing them.

The Computer-Integrated Manufacturing Open-System Architecture (CIMOSA) (AMICE Consortium, 1993) aims at integrating enterprise operations by means of efficient information exchange within the enterprise. CIMOSA models enterprises using four perspectives: the *function view* describes the functional structure required to satisfy the objectives of an enterprise and related control structures; the *information view* describes the information required by each function; the *resource view* describes the resources and their relations to functional and control structures; and the *organization view* describes the responsibilities assigned to individuals for functional and control structures. The Generalised Enterprise Reference Architecture and Methodology (GERAM) (Bernus and Nemes, 1996) defines a set of concepts for designing and maintaining enterprises during their entire life-history spanning from products to enterprise integration and strategic enterprise management. This framework identifies basic concepts used 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 business processes and their supporting technologies.

Among proposals from the multi-agent systems domain, OperA (Dignum, 2004) aims at designing 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 3-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 to roles through a social contract; finally, the *interaction model* describes the society agents interactions by the means of interaction contracts. The OperA framework is supported by a language based on deontic temporal logic that provides a formal framework and integrated semantics at all three levels of society specification. MOISE+ (Hübner et al., 2002) focuses on the structure and functionalities 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 groups to model the individual, social, and collective structural levels of organizations. The functional specification is based on the concepts of missions and global plans. The deontic specification then links the structural specification to functional specification in terms of permissions and obligations.

The Tropos methodology (Bresciani et al., 2004) is an agent-oriented software engineering methodology intended to support all analysis and design activities in the software development process. The methodology consists of five phases, namely Early Requirements, Late Requirements, Architectural Design, Detailed Design, and Implementation. Early Requirements aims at understanding the domain with its stakeholders and their individual and shared goals. Late Requirements focuses on the elicitation of requirements for the system-to-be. Architectural Design specifies the system architecture in terms of a set of interacting software agents. Detailed Design is concerned with the specification of agent capabilities and interaction. Finally, Implementation deals with the production of code from the detailed design specification. Tropos adopts the i* modeling language (Yu, 1996), which allows designers to model the organizational environment of a system and the system itself. This language offers primitive concepts such as actor, goal, plan, resource, as well as social dependency relationships between two actors. The modeling framework of i* includes strategic dependency models for describing the network of inter-dependencies among actors, as well as strategic rationale models for describing and supporting the reasoning of each actor vis-a-vis other actors.

Among proposals for Semantic Web, we note the Descriptive Ontology for Linguistic and Cognitive Engineering (DOLCE) (Masolo et al., 2004). DOLCE aims to capture ontological categories that underlie natural language and human common sense. This ontology uses three main entities for modeling organizational settings: organizations, norms and roles. Norms describe the structure and purposes of an organization by identifying its main concerns and the behavior of its agents. The link between agents and norms is represented in terms of roles.

In the realm of security and privacy modeling, we find sophisticated proposals such as XACML (OASIS, 2005), EPAL (Ashley et al., 2003), and P3P (Cranor et al.,

2002). XACML is an OASIS standard supporting both an access control policy language and an access control decision language. XACML defines schemes for the specification of both a context and access control policies. An EPAL policy is a set of privacy rules that includes a data user, an action, a data category, and a purpose with conditions and obligations. On the other hand, P3P aims at formalizing privacy statements that are published by an enterprise. Its goal is to define a machine-readable equivalent for the human readable privacy promises that are published as a privacy statement on a web page. Unlike XACML and EPAL, P3P defines a global terminology that can be used to describe privacy policies for an enterprise. However, these standards do not address issues of design: the system administrator must manually decide which is the right policy to protect the information system he is responsible for. Moreover, these proposals do not provide facilities for modeling the structure of an organization together with organizational goals. Accordingly, it is not possible to verify whether a given policy is consistent with the functionalities of the system.

Requirements Engineering usually treats security as a non-functional requirement (Chung et al., 2000). Non-functional requirements introduce quality characteristics, but they also represent constraints under which the system must operate (Sommerville, 2001). Although system designers have recognized the need to integrate most of the non-functional requirements, such as reliability and performance, into the software development process (Dardenne et al., 1993), security requirements are identified after the definition of the functional design. This attitude may lead to generating serious design challenges that usually translate into software vulnerabilities or serious organizational blunders.

Security needs are generically expressed by organizational security policies. An organization defines high-level policies about security with respect to its strategic objectives and its organizational structure. Such policies have to be mapped to the specific functionalities of their information systems. Without an explicit model of the organization and the trust relationships among its components it can be result particularly complex to find the reasons that have motivated their introduction (Lampson, 2004). For instance, ignoring trust concerns seriously affects the effectiveness of security measures imposed on a system. For instance, system designers may not introduce security measures since they may implicitly assume trust relationships among users that are in fact not there in the domain. Alternatively, system designers may introduce expensive mechanisms for protecting a trusted system that has not been perceived as such by designers.

The purpose of this chapter is to define a novel ontology supporting the integration of security and requirements engineering during early phases of system development. Such an ontology is intended to aid designers in understanding why security mechanisms such as authentication, access control, or back ups are necessary, and once they are selected, what are the trade-offs from the standpoint of corporate missions. Although there have been several proposals for modeling security features, what is still missing are models that focus on high-level security concerns without forcing designers to immediately get down to security mechanisms. For instance, Jürjens (2004) proposed UMLsec for modeling security related features, such as confidentiality and access control. Basin et al. (2006) proposed an UML-based modeling language, SecureUML. Their approach is focused on modeling access

control policies and integrating them into a model-driven software development process. McDermott and Fox (1999) adapt use cases to capture and analyze security requirements, and they 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 one of the stakeholders of the system. Guttorm and Opdahl (2005) define misuse cases, the converse of UML use cases, which describe uses that the system should not allow.

3 A Running Example

A major source of vulnerabilities is due to the presence of conflicts and loopholes at the interface between an IT system and its operational environment. Only analyzing the system from an organizational perspective designers can identify appropriate security solutions.

An application domain where such issues are prominent is the banking domain. Banks, by their very nature, have to enforce security in the context of distributed control and responsibility, also evolving services and infrastructures. Protection measures, such as access control policies, separation of duties, auditing, non-repudiation action, digital signatures, all need to be considered and applied to comply with security and legal requirements besides functional requirements for a system-to-be.

In this chapter, we focus on a banking scenario and more specifically on loan process in the context of which activities take place and assignment of rights, roles, and tasks need to be carefully considered from a security perspective. In this scenario, we are going to emphasize the necessity of preventing frauds, preserving data integrity, and protecting customer privacy rights.

Si*: A LANGUAGE FOR SRE

The definition of a modeling language for designing secure socio-technical systems includes the definition of primitive concepts for modeling organizational and security concerns, as well as the logical formalization of such primitives. Our language, Si* (Secure i*), is based on the i* ontology (Yu, 1996), where specifications employ basic primitives such as “actor”, “role”, “goal”, “task”, “resource”, and “social relationships between actors”.

Actors and their specializations

An actor is an active entity that has strategic goals and performs actions to achieve them. Actors can be decomposed into sub-units for modeling the internal structure of organizations. Complex social actors can be modeled using two types of sub-units: agents and roles. An *agent* is an actor with concrete, physical manifestations. The term agent can be used to refer to human as well as software agents and organizations. A *role* is the abstract characterization of the behavior of a social actor within some specialized context. Figure 1 shows the graphical representation of actors and their specializations.

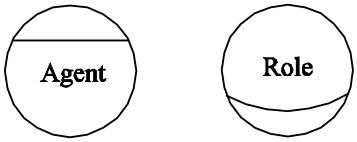

Figure 1 Si* graphical representation of agents and roles

An agent is said to *play* a role. The play relation is similar in the intuition to the user-role assignment of the RBAC approach (Sandhu et al., 1996). According to such an approach, an agent inherits the properties of the roles he plays. Agents and roles can be further analyzed by decomposing them using the relation *is part of*. For instance, this relation can be used to identify the member of an organization as well as the sub-components of a software agent.

Si* provides support for modeling role hierarchies based on the concepts of specialization and supervision. 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. The basic idea underlying supervision is that, if a role supervises another role, the first is responsible for the behavior of the latter and has the capabilities to control and evaluate the latter's work. This concept is used to build the supervision hierarchy (Figure 3), whereas the specialization hierarchy is built using the ISA relation (Figure 2).

Example 1 *The director of a bank is responsible for the correct delivery of the services offered by bank itself. The director cannot perform all such services by himself, and so appoints managers and clerks (e.g., pre-processing clerks and post-processing clerks) to perform some of the tasks he is responsible for. If services are not provided in compliance with bank policies, he is personally liable. Thereby, the director has good reasons to check and evaluate the behavior of subordinate roles. Figures 2 and 3 represent the roles presented above and the relations between them.*

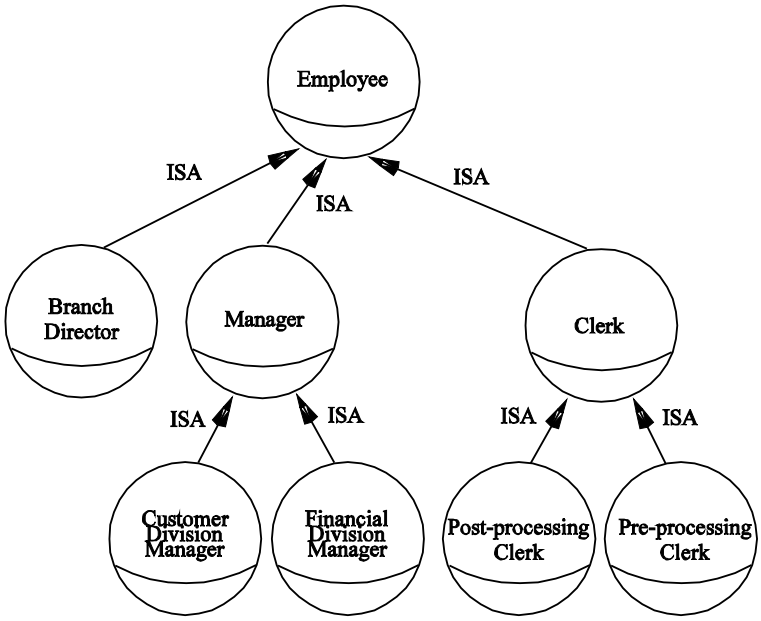

Figure 2 Specialization Hierarchy

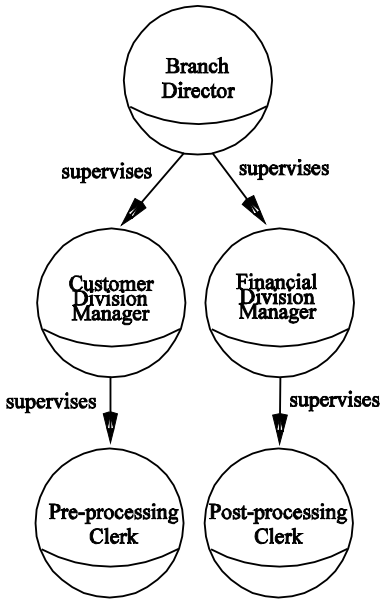

Figure 3 Supervision Hierarchy

Goals, tasks and resources

A *goal* represents a strategic interest of an actor. Si*, as well as i*, differentiates between hard (only goals hereafter) and soft goals. The latter have no clear definition or criteria for deciding whether they are satisfied or not, and are typically used to model non-functional requirements. According to (Chung et al., 2000), the different nature of fulfillment is underlined by saying that goals are *satisfied*, while softgoals are *satisficed*.

Goals can be fulfilled by means of tasks or resources. A *task* represents a particular course of actions that produces a desired effect. A task can be executed in order to satisfy a goal or satisfice a softgoal. A *resource* represents a physical or an informational entity without intentionality. A resource can be consumed or produced by a task. Figure 4 depicts the graphical representation of goals, softgoals, tasks, and resources.

Figure 4 Si* graphical representation of goal, softgoal, task, and resource

Si* is based on the idea of building a model of the system that is incrementally refined and extended. Goal modeling consists of refining goals and eliciting new social relationships among actors. Goals are analyzed from the perspective of single actors using three techniques, namely *AND/OR decomposition*, *contribution analysis*, and *means-end analysis*. AND/OR decomposition combines AND and OR refinements of a root goal into subgoals, modeling a finer goal structure. In essence, AND-decomposition is used to define the process for achieving a goal, while OR-decomposition defines alternatives for achieving a goal. Contribution analysis identifies goals and tasks that contribute positively or negatively in the fulfillment of

the goal to be analyzed. Means-end analysis aims at identifying goals, softgoals, tasks, and resources that provide means for achieving a goal.

Example 2 *One of the services offered by the bank is to* offer loans*. The provisioning of such a service contributes to* increase bank profits*. The bank AND-decomposes* offer loans *into* identify customers*,* manage the loan process*,* sell the loan*. These subgoals can be further decomposed until a plan to fulfill them is identified. For instance,* getting customer data *can be achieved by executing tasks* insert customer identifier *and* retrieve customer data*. Figure 5 shows the goal diagram derived applying goal analysis to* offer loans*.*

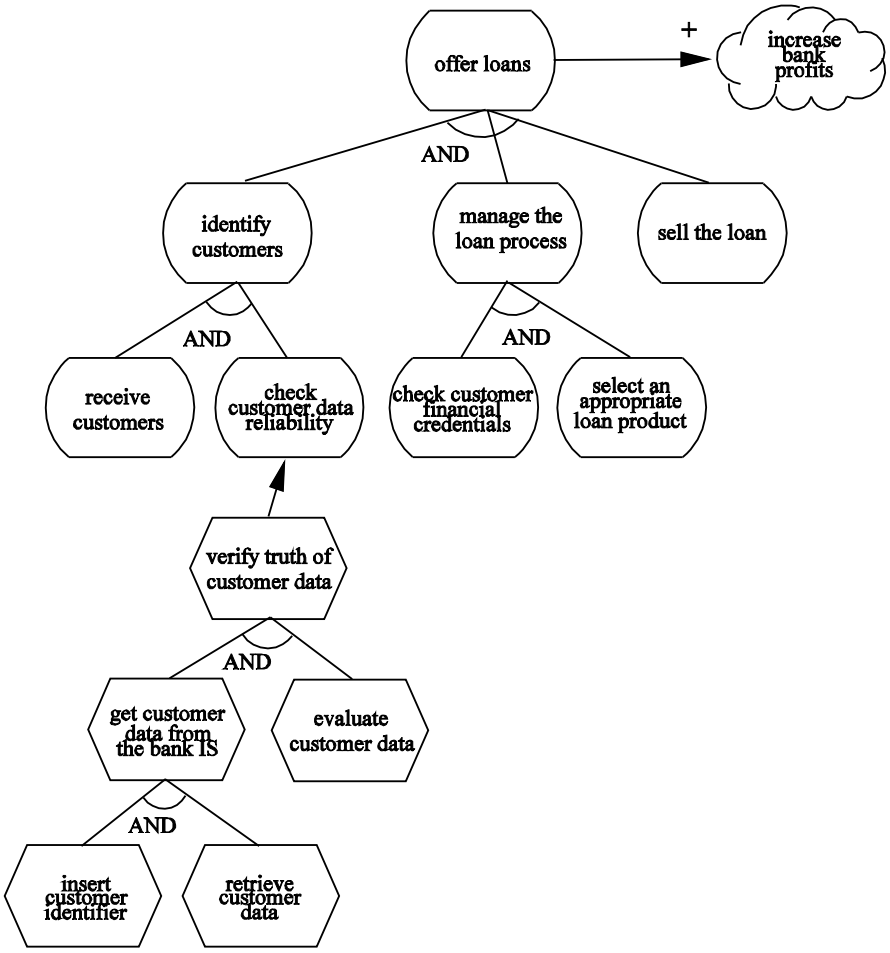

Figure 5 Goal diagram

Objectives, Entitlements, and Capabilities

The first intuition in modeling security aspects of information systems is to distinguish between actors who want access to a resource, fulfillment of a goal or execution of a task, from actors who have the capabilities to do any of the above, and – last but not least – actors who are entitled to do any of the above. Essentially, every actor is defined along with a set of objectives, capabilities, and entitlements.

Objectives, entitlements and capabilities of actors are modeled through relations between actors and services, namely *request*, *own*, and *provide*.

Request indicates that an actor intends to achieve a goal, execute a task, or requires a resource.

Own indicates that an actor is the legitimate "owner" of a goal, a task, or a resource. The basic idea is that an owner has full authority concerning access and disposition over his entitlements.

Provide indicates that the actor has the capability to achieve a goal, execute a task, or deliver a resource.

The distinction between being entitled and providing allows us to model situations where the actor that has the capabilities to fulfill a goal is different from the one that has the permission to do it.

Example 3 *According to data protection legislation, a customer is entitled to control the use of his personal data. The pre-processing clerk is appointed to identify customers. Thereby, he needs to access customer information to achieve his duties. However, he does not directly interact with the customer, but he retrieves such data from the bank IT system. Thus, the bank should seek the consent of the customer for granting access to the customer's data to all employees assigned to him.*

Relations *request*, *own*, and *provide* are graphically represented as edges between an actor and a service, labeled by **R**, **O** and **P**, respectively.

Trust and Delegation

Si* supports the notion of *delegation* in order to model the transfer of entitlements and responsibilities from an actor to another. Thus, delegation is a ternary relation among two actors (the *delegator* and the *delegatee*) and a goal, task or resource (the *delegatum*).

Example 4 *A pre-processing clerk is interested in gathering customer data, for which he depends on the bank IT system. The customer delegates the permission to provide his data to the bank IT system on condition that they are not disclosed to third parties.*

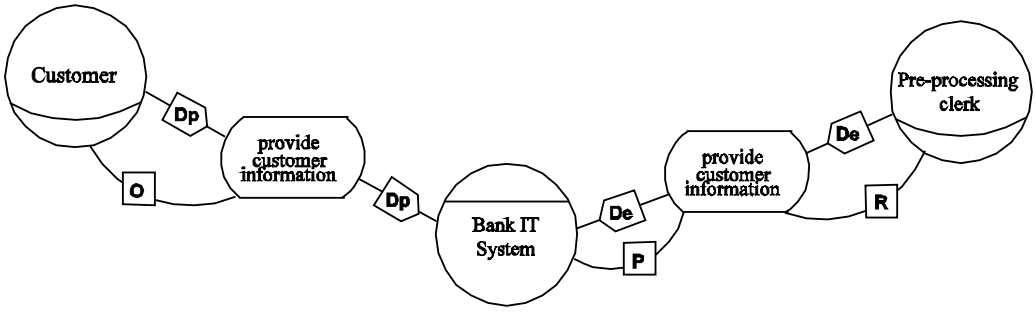

Figure 6 Delegation

In this scenario (Figure 6), there is a difference of relationship between the pre-processing clerk and the bank IT system and between the customer and the bank IT system. This difference is based on the type of delegation used in the two relationships. Thereby, we introduce a conceptual refinement of delegation, that allows us to capture and model important security facets.

Delegation of execution indicates that one actor delegates to other actors the responsibility to achieve a goal, execute a task, or deliver a resource. This would be matched, for instance, by a call to an external procedure. As consequence, the delegatee is responsible for the achievement of the goal, execution of the task, or delivery of the resource.

Delegation of permission indicates that one actor delegates to other actors the permission to achieve a goal, execute a task, or use a resource. This would be matched by issuing a delegation certificate, such as digital credential or a letter. As consequence, the delegatee is entitled to achieve the goal, execute the task, or use the resource.

In the graphical representation of Figure 6 we represent these relationships as edges respectively labeled **De** and **Dp**.

Example 5 *The customer* delegates *the* permission *to the bank IT system to provide only information relevant for the required service. On the other hand, the pre-processing clerk, who wants customer data,* delegates *the* execution *of his goal to the bank IT system. According to the pre-processing clerk, the bank IT system should provide the required information. He is not interested in what the bank IT system does with the customer consent, apart from getting his information. The clerk's major concern would be that tasks are delegated to people that can actually do them, whereas the customer would be concerned that his data are given to people who will not misuse the permissions they have acquired.*

Further, we want to separate the concepts of trust and delegation, as we might need to model systems where some actors must delegate permission or execution to untrusted actors. Trust represents the willingness to accept vulnerability based on positive expectations about the behavior of another actor (Mayer et al,. 1995; Rousseau et al., 1998). It is related to belief in honesty, trustfulness, competence, and reliability (Castelfranchi and Falcone, 1998) and it is used to build collaboration between humans and organizations (Axelrod, 1984). Trust is an important aspect for making decisions on security since it allows to economize on information processing and protection mechanisms.

Similarly to delegation, we represent trust as a ternary relation among two actors (the *trustor* and the *trustee*) and a goal, a task or a resource. The object around which the trust relationship centers is called *trustum*. Also in this case it is convenient to have a suitable distinction for trust in managing permission and trust in managing execution.

Trust of execution indicates the belief of one actor that the trustee will achieve the goal, perform the task, or furnish the resource.

Trust of permission indicates the belief of one actor that the trustee will not misuse the goal, task, or resource.

These relationships are graphically represented as edges respectively labeled **Te** and **Tp**.

A FORMAL ONTOLOGY

To define a formal semantics for the new primitives, we use the Answer Set Programming (ASP) paradigm (Leone et al., 2006). The ASP paradigm is based on the concepts of facts and rules expressed as Horn clauses and evaluated using the stable model semantics.[ii] Facts are atomic statements representing the extensional description of the system. Rules can be axioms or properties: axioms are used to complete the extensional description of the system, whereas properties correspond to integrity constraints and are used to verify requirements consistency.

Predicates

Our setting distinguishes two types of predicates: intensional and extensional. Extensional predicates (Table 1) correspond to the edges and circles drawn by the requirements engineer during the modeling phase and are used to formalize the intuitive description of the system. Intensional predicates (Table 2) are determined with the help of rules by the reasoning system.

Extensional Predicates

For an automatic and precise analysis of requirements, graphical diagrams need to be translated in formal specifications. This has spurred us to define an extensional predicate for each primitive concept. Next, this set of predicates is presented and a summary is given in Table 1.

Type Predicates
service(Service:s)
goal(Goal:g)
task(Task:t)
resource(Resource:r)
actor(Actor:x)
agent(Agent:a)
role(Role:p)
Goal Analysis
AND_decomp(Service:s,Service:s1,Service:s2)
OR_decomp(Service:s,Service:s1,Service:s2)
pos_contribution(Service:s1,Service:s2)
neg_contribution(Service:s1,Service:s2)
means_end(Service:s1,Service:s2)
Association Relations
play(Agent:a,Role:p)
is_a(Role:p,Role:q)
supervise(Role:p,Role:q)
is_part_of(Actor:x,Actor:y)
Actor Properties
request(Actor:x,Service:s)
own(Actor:x,Service:s)
provide(Actor:x,Service:s)
Delegation and Trust
delegate(perm,x,y,s)
delegate(exec,x,y,s)

trust(perm,x,y,s)
trust(exec,x,y,s)

Table 1 Extensional predicates

- *Type Predicates*: The unary predicates *goal*, *task* and *resource* are used respectively for identifying goals, tasks, and resources. For sake of compactness, we will use the unary predicate *service* when it is not necessary to distinguish among goals, tasks, and resources. We shall use letters *S*, *G*, *T* and *R* possibly with indexes as variables ranging over services, goals, tasks and resources, respectively. The unary predicates *agent* and *role* are used respectively for identifying agents, and roles. For sake of compactness, we introduce the unary predicate *acto*r when is not necessary to distinguish among them. We shall use letters *X*, *Y* and *Z* as variable to indicate generic actor, *A*, *B* and *C* as variables to indicate agents, and *P*, *Q* and *V* as variables to indicate roles.
- *Goal Analysis*: Predicates *AND_decomp* and *OR_decomp* are used to model AND- and OR-decomposition, respectively. Predicates *pos_contribution* and *neg_contribution* are used to model positive and negative contribution, respectively. Finally, *means_end* states that a service provides means for achieving a goal with respect to the perspective of an actor.
- *Association Relations*: Predicate *play* identifies the roles played by an agent. Predicate *is_a* is used to build specialization hierarchies, whereas *supervise* is used to build supervision hierarchies. Finally, *is_part_of* identifies the sub-components of an actor.
- *Actor Properties*: Predicate *request* identifies the objectives of actors, *provide* the capabilities of actors, and *own* the legitimate owner of services.
- *Delegation and Trust*: Predicates *delegate(perm,x,y,s)* and *delegate(exec,x,y,s)* correspond to delegation of permission and delegation of execution, respectively. Predicates *trust(perm,x,y,s)* and *trust(exec,x,y,s)* correspond to trust of permission and trust of execution, respectively.

Intensional Predicates

The intuitive description of the system is not sufficient for an accurate verification of the system (Giorgini et al., 2006). To derive the right conclusions, such a description is completed using rules. To distinguish the relations drawn by the requirements engineer from the ones derived by the system, we introduce a set of intensional predicates (Table 2). Next, we present such predicates.

Goal Analysis
subservice(Service:s1,Service:s2) *AND_subservice(Service:s1,Service:s2)* *OR_subservice(Service:s1,Service:s2)*
Actor Properties
aim(Actor:x,Service:s) *has_perm(Actor:x,Service:s)*
Trust
trustChain(perm,Actor:x,Actor:y,Service:s) *trustChain(exec,Actor:x,Actor:y,Service:s)*
Confidence and Need-to-Know
in_charge(Actor:x,Service:s)

fulfill(Actor:x,Service:s)
can_satisfy(Actor:x,Service:s)
can_execute(Actor:x,Service:s)
confident(satisfy,Actor:x,Service:s)
confident(execute,Actor:x,Service:s)
confident(owner,Actor:x,Service:s)
need_to_have_perm(Actor:x,Service:s)

Table 2: Intensional Predicates

- *Goal Analysis*: These predicates identify the relations among services in terms of subparts. Predicates *subservice*, *OR_subservice* and *AND_subservice* respectively identifies a subservice, OR-subservice and AND-subservice of a service. More specific predicates should be introduced for goal, task and resource decomposition.
- *Actor Properties*: Predicate *aim* identifies direct and indirect objectives of actors and *has_perm* identifies direct and indirect entitlements of actors.
- *Trust*: Trust relations can be combined to build trust chains. In particular, *trustChain(perm,x,y,s)* and *trustChain(perm,x,y,s)* chains of trust of permission and trust of execution, respectively.
- *In charge and fulfill*: Predicate *in_charge* identifies actors who take care of the final delivery of a service and *fulfill* identifies actors who are actually willing to deliver a service.
- *Confidence of execution*: This set of predicates is used to capture the notion of confidence from the requester's perspective. Predicate *can_satisfy* identifies actors who delegate their objectives to actors who have the capabilities to fulfill them. Predicate *can_execute* identifies actors who delegate their objectives to actors who will fulfill them. *confident(satisfy,x,s)* identifies actors confident that a service can be satisfied. *confident(execute,x,s)* identifies actors confident that a service will be fulfilled. This is the case if an actor knows that all delegations have been done to trusted actors and that the actor, who will ultimately deliver the service, has permission to do so.
- *Confidence of entitlements*: From the viewpoint of the owner, confidence means that the owner is confident that the permission that he has delegated will not be misused. Thereby, *confident(owner,x,s)* holds if an actor is confident that the permission on his entitlements is granted only to trusted actors.
- *Need-to-Know*: Current privacy and data protection legislation requires that information is unavailable to actors except those who need legitimately to know (need-to-know principle). Essentially, this corresponds to the desire of owners to delegate permissions to providers only if the latter actually do need the permission. Predicate *need_to_have_perm* is used to capture this idea.

Axioms

This section describes the axioms that define the semantics underlying Si*. They are used to complete the extensional description of the system.[iii]

Trust

Table 3 presents the axioms for propagating trust relations along chains and service refinement.

Trust	
T1	*trustChain(exec,X,Y,S)←trust(exec,X,Y,S)*
T2	*trustChain(exec,X,Y,S)←trust(exec,X,Z,S)∧trustChain(exec,Z,Y,S)*
T3	*trustChain(exec,X,Y,S1)←subservice(S,S1)∧trustChain(exec,X,Y,S)*
T4	*trustChain(perm,X,Y,S)←trust(perm,X,Y,S)*
T5	*trustChain(perm,X,Y,S)←trust(perm,X,Z,S)∧trustChain(perm,Z,Y,S)*
T6	*trustChain(perm,X,Y,S)←subservice(S,S1)∧trustChain(perm,X,Y,S1)*

Table 3: Trust Propagation

- *Trust* (T1-6) T1-2 are used to build trust chains for execution. T3 propagates trust relationships from a service to its parts. T4-5 are used to build trust chains for permission. T6 propagates trust along service refinements. If an actor trusts that another will not overstep the set of actions required to fulfill a part of a service, then the first can trust the last will not overstep the set of actions required to fulfill the service. Thereby, trust of permission flows bottom-up with respect to goal refinements.

Fulfillment, Confidence, and Need-to-Know

Tables 4 and 5 present the set of axioms for identifying entitlements and responsibilities of actors; also, actors who will fulfill services and actors who are confident that their objectives will be fulfilled and their entitlements will not misused.

Aims	
AP1	*aim(X,S)←request(X,S)*
AP2	*aim(X,S) ←delegate(exec,Y,X,S)∧aim(Y,S)*
AP3	*aim(X,S)←subservice(S1,S)∧aim(Y,S)*
Has permission	
AP4	*has_perm(X,S)←own(X,S)*
AP5	*has_perm(X,S) ←delegate(perm,Y,X,S)∧has_perm(Y,S)*
AP6	*has_perm(X,S)←subservice(S1,S)∧has_perm(Y,S)*
In charge	
AP7	*in_charge(X,S)←aim(X,S)∧provide(X,S)*
Fulfill	
AP8	*fulfill(X,S)←in_charge(X,S)∧has_perm(X,S)*
Can satisfy	
AP9	*can_satisfy(X,S)←in_charge(X,S)*
AP10	*can_satisfy(X,S)←delegate(exec,X,Y,S)∧can_satisfy(Y,S)*
AP11	*can_satisfy(X,S)←OR_subservice(S1,S)∧can_satisfy(X,S1)*
AP12	*can_satisfy(X,S)←AND_decomp(S,S1,S2)∧can_satisfy(X,S1)∧can_satisfy(X,S2)*
Can execute	
AP13	*can_execute(X,S)←fulfill(X,S)*
AP14	*can_execute(X,S)←delegate(exec,X,Y,S)∧can_execute(Y,S)*
AP15	*can_execute(X,S)←OR_subservice(S1,S)∧can_execute(X,S1)*
AP16	*can_execute(X,S)←AND_decomp(S,S1,S2)∧can_execute(X,S1)∧can_execute(X,S2)*

Table 4: Entitlements and Objectives Transfer and Fulfillment

- *Aim* (AP1-3) AP1 states that if an actor requests a service fulfilled, he aims its fulfillment. AP2 states that if an actor requires a service delivered and delegates its execution to another actor, the service becomes an objective of the delegatee. Finally, AP3 propagates objectives through service refinement.
- *Has permission* (AP4-6) The owner of a service has full authority concerning access and disposition of it. Thus, AP4 states that if an actor owns a service, he is entitled to deliver it. AP5 states that if an actor is entitled to deliver a service and delegates the permission to another actor, the delegatee is entitled to deliver the service. Finally, AP6 propagates entitlements through service refinement.
- *In charge* (AP7) An actor will take charge of the fulfillment of a service if he has the capabilities to fulfill it and it belongs to his objectives.
- *Fulfill* (AP8) An actor will fulfill a service if he has taken charge of its fulfillment and has the permission to fulfill it.
- *Can satisfy* (AP9-12) An actor can satisfy his objectives if either he has taken charge of them (AP9) or has delegated them to someone who can satisfy them (AP10). Service decompositions are accounted for through axioms AP11-12. If an actor can satisfy at least one of the OR-subservices of a service, then he can satisfy the root service. Dual axiom holds for AND-decompositions.
- *Can execute* (AP13-16) These axioms is used to identify actors that actually can deliver a service by combining execution with permission. An actor can fulfill his objectives if either he will fulfill them directly (AP13) or has delegated its execution to someone who can execute them (AP14). Service decompositions are accounted for through axioms AP15-16. If an actor can execute at least one of the OR-subservices of a service, then he can execute the root service. Dual axiom holds for AND-decompositions.

Confident of satisfaction	
AP17	$confident(satisfy,X,S) \leftarrow in\_charge(X,S)$
AP18	$confident(satisfy,X,S) \leftarrow delegate(exec,X,Y,S) \wedge trustChain(exec,X,Y,S) \wedge confident(satisfy,X,S)$
AP19	$confident(satisfy,X,S) \leftarrow OR\_subservice(S1,S) \wedge confident(satisfy,X,S1)$
AP20	$confident(satisfy,X,S) \leftarrow AND\_decomp(S,S1,S2) \wedge confident(satisfy,X,S1) \wedge confident(satisfy,X,S1)$
Confident of execution	
AP21	$confident(execute,X,S) \leftarrow fulfill(X,S)$
AP22	$confident(execute,X,S) \leftarrow delegate(exec,X,Y,S) \wedge trustChain(exec,X,Y,S) \wedge confident(execute,X,S)$
AP23	$confident(execute,X,S) \leftarrow OR\_subservice(S1,S) \wedge confident(execute,X,S1)$
AP24	$confident(execute,X,S) \leftarrow AND\_decomp(S,S1,S2) \wedge confident(execute,X,S1) \wedge confident(execute,X,S1)$
Confident of entitlements	
AP24	$confident(owner,X,S) \leftarrow owns(X,S) \wedge not\ diffident(X,S)$
AP26	$diffident(X,S) \leftarrow delegate(exec,X,Y,S) \wedge not\ trustChain(perm,X,Y,S)$
AP27	$diffident(X,S) \leftarrow delegate(exec,X,Y,S) \wedge diffident(X,S)$
AP28	$diffident(X,S) \leftarrow subservice(S1,S) \wedge diffident(X,S)$
Need to know	
AP29	$need\_to\_have\_perm(X,S) \leftarrow in\_charge(X,S)$

AP30	*need_to_have_perm(X,S)←delegate(perm,X,Y,S)∧not other_delegater(X,Y,S)∧ need_to_have_perm(Y,S)*
AP31	*other_delegater(X,Y,S)←delegate(perm,X,Y,S)∧delegate(perm,Z,Y,S)∧ need_to_have_perm(Z,S)∧X≠Z*

Table 5: Confidence and Need-to-Know

- *Confidence of satisfaction* (AP17-20) An actor is confident that its objectives will be satisfied if he takes care of them (AP17) or he has delegated their execution to trusted actors (AP18). Axioms AP19-20 specify how confidence of satisfaction is propagated upwards along service decomposition.
- *Confidence of execution* (AP21-24) An actor is confident to fulfill his objectives if he fulfills them by himself (AP21) or he has delegated their execution to trusted actors (AP22). Axioms AP23-24 propagate confidence of execution upwards along service decomposition.
- *Confidence of entitlements* (AP25-28) An owner is confident, if there is no likely misuse of his permission. It can be seen that there is an intrinsic double negation in the statement. We model it using a predicate *diffident*. A delegating agent is diffident, if the delegation is being done to an untrusted agent (AP26) or if the delegatee could be diffident himself (AP27). AP28 propagates diffidence upwards along service decomposition.
- *Need to Know* (AP29-31) These axioms defines the semantics of intensional predicates that are necessary to analyze *need-to-know* properties. These axioms also capture the possibility of having alternate paths of permission delegations through predicate *other_delegater*. In this case the formal analysis will not yield one model but multiple models in which only one path of delegation is labeled by the need-to-have property and the others are not. Essentially, AP30-31 introduce non-determinism, so they make search and verification harder.

ANALYSIS AND VERIFICATION

The suggested primitives were sufficient to deal with most of the security organizational requirements we encountered. For instance, it has been shown that Si* is able to cope with the complexity of a real ISO-17799-like case study (Massacci et al., 2005). Security requirements are verified using *properties*. Such properties are defined in form of patterns that have to be checked. In ASP, they are represented as constraints that a good design should satisfy. If these features are not consistent, vulnerabilities may occur in the implementation of the system-to-be. Table 6 presents the basic set of properties.

Authorization	
Pro1	*←delegate(perm,X,Y,S)∧not trustChain(perm,X,Y,S)*
Pro2	*←delegate(perm,X,Y,S)∧not has_perm(X,S)*
Pro3	*←own(X,S)∧not confident(owner,X,S)*
Availability	
Pro4	*←delegate(exec,X,Y,S)∧not trustChain(exec,X,Y,S)*
Pro5	*←request(X,S)∧not can_satisfy(X,S)*
Pro6	*←request(X,S)∧not can_execute(X,S)*
Pro7	*←request(X,S)∧not confident(satisfy,X,S)*

Pro8	←*request(X,S)*∧*not confident(execute,X,S)*
Pro9	←*need_to_have_perm(X,S)*∧*not has_perm(X,S)*
Privacy	
Pro10	←*has_perm(X,S)*∧*not need_to_have_perm(X,S)*

Table 6: Security Properties

- *Authorization* (Pro1-3) Pro1 is used to detect untrusted delegations of permission. Pro2 verifies whether an actor who delegates the permission to deliver a service is entitled to do it. Pro3 verifies that the owner of the service has to be confident to give the service only to trusted actors.
- *Availability* (Pro4-9) Pro4 is used to detect untrusted delegations of execution. Pro5-6 check if actors can satisfy and execute the required services. Pro7-8 verify whether requesters are confident to satisfy and execute required services, respectively. Pro9 verifies if actors have the permission necessary to perform their duties.
- *Privacy* (Pro10) Pro10 verifies that actors, who have the permission on a service, actually need such permission.

CONCLUSION

This chapter has proposed an ontology intended to model security at an organizational level. The proposed concepts proved up to the challenge, and revealed a number of pitfalls, especially when formal analysis techniques were applied (Massacci and Zannone, 2006).

We are currently extending the ontology to capture behavioral aspects of the system. This extension has two implications. On one hand, it allows system designers to capture more sophisticated security properties. On the other hand, such concepts support the (semi-)automatic derivation of business processes from the requirements model.

Another direction under investigation involves the enrichment of the Si* ontology with concepts necessary for capturing privacy concerns. According to existing privacy legislations in many countries (e.g., the US Privacy Act and the EU Privacy Directive), privacy is mainly maintained by controlling the usage of information. This requires that information be linked to the functional requirements of the original application. Following this trend, researchers have recently proposed frameworks for specifying and enforcing privacy policies. However, they do not support policy writers in the analysis of organizational requirements and leave them to manually define privacy policies. Our objective is to bridge the gap between the requirements analysis and policy specification by deriving privacy policies directly from the requirements model.

ACKNOWLEDGMENTS

We thank Nicola Guarino and ISTC-CNR Laboratory for Applied Ontology in Trento for many useful discussions. This work was partly supported by the projects FIRB-TOCAI, IST-FP6-FET-IP-SENSORIA, IST-FP6-IP-SERENITY, and PAT-MOSTRO.

REFERENCES

AMICE Consortium (1993). *Open System Architecture for CIM*. Springer-Verlag.

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

Ashley, P., Hada, S., Karjoth, G., Powers, C., & Schunter, M. (2003). Enterprise Privacy Authorization Language (EPAL 1.2). W3C Recommendation.

Axelrod, R. (1984). *The Evolution of Cooperation*. Basic Books.

Basin, D., Doser, J., & Lodderstedt, T. (2006). Model Driven Security: from UML Models to Access Control Infrastructures. *ACM Transactions on Software Engineering and Methodology*, 15(1):39–91.

Bernus, P., & Nemes, L. (1996). A Framework to Define a Generic Enterprise Reference Architecture and Methodology. *Computer Integrated Manufacturing Systems*, 9(3):179–191.

Bresciani, P., Giorgini, P., Giunchiglia, F., Mylopoulos, J., & Perini, A. (2004). TROPOS: An Agent-Oriented Software Development Methodology. *Autonomous Agents and Multi-Agent Systems*, 8(3):203–236.

Bryce, M., & Associates (2006). PRIDE-EEM Enterprise Engineering Methodology. From http://www.phmainstreet.com/mba/pride/eemeth.htm.

Castelfranchi, C., & Falcone, R. (1998). Principles of trust for MAS: Cognitive anatomy, social importance and quantification. In *International Conference on Multi-Agent Systems*, pages 72–79. IEEE Press.

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

Cranor, L., Langheinrich, M., Marchiori, M., & Reagle, J. (2002). The Platform for Privacy Preferences 1.0 (P3P1.0) Specification. W3C Recommendation.

Dardenne, A., van Lamsweerde, A., & Fickas, S. (1993). Goal-directed Requirements Acquisition. *Science of Computer Programming*, 20:3–50.

Dignum, V. (2004). *A model for organizational interaction: based on agents, founded in logic*. PhD thesis, Universiteit Utrecht.

Emery, F. E., & Trist E. L. (1960). Socio-technical systems. In *Management Sciences: Models and Techniques*, volume 2, (pp 83–97). Pergamon Press

Giorgini, P., Massacci, F., Mylopoulos, J., & Zannone, N. (2006). Requirements Engineering for Trust Management: Model, Methodology, and Reasoning. *International Journal of Information Security*, 5(4):257–274.

Giorgini, P., Massacci, F., & Zannone, N. (2005). Security and Trust Requirements Engineering. In *FOSAD III*, LNCS 3655, (pp. 237–272). Springer.

House of Lords (1999). Prince Jefri Bolkiah vs KPMG. 1 All ER 517. From www.parliament.the-stationeryoffice.co.uk.

Hübner, J. F., Sichman, J. S., & Boissier, O. (2002). A Model for the Structural, Functional, and Deontic Specification of Organizations in Multiagent Systems. In *Brazilian Symposium on Artificial Intelligence*, pages 118–128. Springer.

Jürjens, J. (2004). *Secure Systems Development with UML*. Springer-Verlag.

Lampson, B. W. (2004). Computer Security in the Real World. *Computer*, 37(6):37–46.

Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S., & Scarcello, F. (2006). The DLV System for Knowledge Representation and Reasoning. *ACM Transactions on Computational Logic*, 7(3):499–562.

Liu, L., Yu, E. S. K., & Mylopoulos, J. (2003). Security and Privacy Requirements Analysis within a Social Setting. In *IEEE International Requirements Engineering Conference*, pages 151–161. IEEE Press.

Masolo, C., Vieu, L., Bottazzi, E., Catenacci, C., Ferrario, R., Gangemi, A., , & Guarino, N. (2004). Social roles and their descriptions. In *Conference on the Principles of Knowledge Representation and Reasoning*, pages 267–277. AAAI Press.

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

Massacci, F., & Zannone, N. (2006). *Detecting Conflicts between Functional and Security Requirements with Secure Tropos: John Rusnak and the Allied Irish Bank* (Technical Report DIT-06-002). University of Trento.

Mayer, R. C., Davis, J. H. , & Schoorman, F. D.. 1995. An integrative model of organizational trust. *Acad. Management Rev*, 20(3):709-734.

McDermott, J., & Fox, C. (1999). Using Abuse Case Models for Security Requirements Analysis. In *Annual Computer Security Applications Conference*, pages 55–66. IEEE Press.

Michaely, R., & Womack, K. L. (1999). Conflict of interest and the credibility of underwriter analyst recommendations. *Review of Financial Studies*, 12(4):653–686.

OASIS (2005). eXtensible Access Control Markup Language (XACML) Version 2.0. OASIS Standard. From http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf.

Rousseau, D. M., Sitkin, S. B., Burt, R. S., & Camerer, C. 1998. Not so different after all: A cross-discipline view of trust. *Acad. Management Rev*. 23(3) 393-404.

Sandhu, R. S., Coyne, E. J., Feinstein, H. L., & Youman, C. E. (1996). Role-based access control models. *Computer*, 29(2):38–47.

Sindre, G., & Opdahl, A. L. (2005). Eliciting security requirements with misuse cases. *Requirements Engineering Journal*, 10(1):34–44.

Sommerville, I. (2001). *Software Engineering*. Addison-Wesley.

Stader, J. (1996). *Results of the Enterprise Project* (Technical Report AIAI-TR-209). University of Edinburgh.

Uschold, M., King, M., Moralee, S., & Zorgios, Y. (1998). The Enterprise Ontology. *Knowledge Engineering Review*, 13(1):31–89.

Yu, E. (1996). *Modelling strategic relationships for process reengineering*. PhD thesis, University of Toronto.

[i] Methodological aspects of this research have been addressed in (Giorgini et al., 2005; Giorgini et al., 2006).
[ii] We assume that the reader is familiar with such concepts. Otherwise see (Leone et al., 2006) for a tutorial.
[iii] We do not present here the axiomatization for the user-role assignment and goal analysis. We refer to (Giorgini et al., 2005) for it.