A Model-Driven Approach for the Specification and Analysis of Access Control Policies*

Fabio Massacci[1] and Nicola Zannone[2]

[1] Department of Information and Communication Technology
University of Trento - Italy
fabio.massacci@unitn.it
[2] Department of Computer Science
University of Toronto - Canada
zannone@cs.toronto.edu

Abstract. The last years have seen the definition of many languages, models and standards tailored to specify and enforce access control policies, but such frameworks do not provide methodological support during the policy specification process. In particular, they do not provide facilities for the analysis of the social context where the system operates.
In this paper we propose a model-driven approach for the specification and analysis of access control policies. We build this framework on top of SI*, a modeling language tailored to capture and analyze functional and security requirements of socio-technical systems. The framework also provides formal mechanisms to assist policy writers and system administrators in the verification of access control policies and of the actual user-permission assignment.

keywords: Security Requirements Engineering, Access Control, Policy Specification

1 Introduction

Access Control is a critical step in securing IT systems as it aims to prevent unauthorized access to sensitive information. An access control system is typically described in three ways: access control policies, models, and mechanisms [38]. Access control policies (the focus of this paper) are sets of rules that specify what users are allowed or not allowed to do in the application domain.

The last years have seen the emergence of languages, models, and standards intended to support policy writers and system administrators in the specification and enforcement of access control policies [4, 8, 14, 24, 33, 35]. Those frameworks however do not provide any methodological support to assist policy writers in capturing the organizational context where the policy will be enforced.

* This work has been partially funded by the EU-IST-IP SERENITY and SENSORIA projects, and by the Canada's NSERC Hyperion project.

The understanding of the social context plays a key role as access control policies must be consistent with the actual organization's practices [32]. Policy writers have to guarantee that the (IT mediated) access control policies in place within the system should protect the system without affecting business continuity. At the same time, they have to prevent the assignment of unnecessary authorizations (the so called *least privilege principle* [37]). Last but not least, they have to ensure that users cannot abuse their position within the organization to gain personal advantages. Thus, several questions arise during the definition of access control policies: "Why does a user need a certain access right?", "Does a user have all permissions he needs to achieve the duties assigned by the organization?", "Does a user have permissions he does not need?", "Can a user abuse his access privileges?", etc.

These issues are critical especially when dealing with sensitive personal information: many countries have issued data protection regulations establishing that the collection and processing of personal data shall be limited to the minimum necessary to achieve the stated purpose [36]. Some proposals [7, 20, 25, 28] have partially answered these issues. For instance, Bertino et al. [7] ensure the least privilege principle by deriving access control policies from functional requirements: users are assigned with the access rights necessary to perform their duties. However, this approach leaves little room for verifying the consistency between security and functional requirements. Even though most policy languages, e.g. XACML [33], are coupled with enforcement mechanisms, very few provide frameworks and tools tailored to analyze the consistency of policies with organizational requirements and to verify the actual assignment of permissions to users.

In this paper, we present a model-driven approach that intends to assist policy writers in the specification and analysis of access control policies and system administrators in making decisions about the assignments of permissions to users. In the development of such a methodology we have taken advantages from both Requirements Engineering (RE) (e.g., [2, 13]) and Trust Management (TM). (e.g., [5, 27]). From RE we get the machinery to model and analyze functional requirements of IT systems and their operational environment. However, RE proposals unlikely address security aspects of organizations. In other words, they focus on what actors should do rather than what actors are authorized to do. TM is orthogonal. It addresses the authorization problem in distributed systems solely. For our purpose, we have chosen the SI* modeling language [30] that integrates concepts from TM, such as permission and its transfer (between actors), into a RE framework. In particular, this language allows the capture and modeling of functional and security aspects of socio-technical systems at the same time.

The first contribution of this paper is a methodological approach for the specifications of access control policies from organizational requirements and their analysis. The consistency of access control policies with functional and security requirements is ensured by verifying the compliance of the requirements models that have generated them with a number of properties of design.

The analysis of security incidents and frauds [3, 21, 34] has revealed that security breaches are often not apparent in policies specified at organizational level, that is, in terms of roles within an organization. To address this issue, we propose to capture security bugs that may be introduced by only modeling organizational requirements by means of a mechanism for instantiating requirements specified at organizational level. This also allows security and system administrators to discard system configurations that may be harmful to the system or to one of the stakeholders of the system through domain-specific constraints (e.g., separation of duties constraints, cardinality of roles, etc.).

Together with a modeling framework, we present a formal framework based on Answer Set Programming (ASP) with value invention [9] to assist policy writers in the specification and analysis of access control policies and system administrators in the user-permission assignment decision making.

In the rest of the paper we provide at first a primer of the SI* modeling language. We then present the process for the specification of access control policies (§3). We propose an approach for the analysis of access control policy at organizational level (§4) and at user level (§5). Access control policies are also analyzed with respect to specificity of the application domain (§6). Next, we propose a formal framework to assist policy writers and system administrators in their task (§7). Finally, we discuss related work (§8) and conclude with some directions for future work (§9).

2 Capturing Organizational Requirements

The SI* modeling language [30] has been proposed to capture security and functional requirements of socio-technical systems. Its main advantage is that it allows the analysis of the organizational environment where the system-to-be will operate and, consequently, it permits to capture not only the *what* and the *how*, but also the *why* security mechanisms have to be introduced in the system.

SI* employs the concepts of agent, role, goal, and task. An *agent* is an active entity with concrete manifestations and is used to model humans as well as software agents and organizations. A *role* is the abstract characterization of the behavior of an active entity within some context. They are graphically represented as circles. Assignments of agents to roles are described by the *play* relation. For the sake of simplicity, in the remainder of the paper we use the term "actor" to indicate agents and roles when it is not necessary to distinguish them.

A *goal* is a state of affairs whose realization is desired by some actor (*objective*), can be realized by some (possibly different) actor (*capability*), or should be authorized by some (possibly different) actor (*entitlement*). Entitlements, capabilities and objectives of actors are modeled through relations between an actor and a goal: *own* indicates that an actor has full authority concerning access and disposition over his entitlement; *provide* indicates that an actor has the capabilities to achieve the goal; and *request* indicates that an actor intends to achieve the goal. A *task* specifies the procedure used to achieve goals. In the

graphical representation, goals and tasks are respectively represented as ovals and hexagons. Own, provide, and request are represented with edges between an actor and a goal labeled by **O**, **P**, and **R**, respectively.

Goals and tasks of the same actor or of different actors are often related to one another in many ways. *AND/OR decomposition* combines AND and OR refinements of a root goal into subgoals, modeling a finer goal structure. Since subgoals are parts of the whole, objectives, entitlements, and capabilities are propagated from a root goal to its subgoals. *Need* relations identify the goals to be achieved in order to achieve another goal. However, neither such goals might be under the control of the actor nor the actor may have the capabilities to achieve them. Therefore, need relations propagate objectives, but not entitlements and capabilities. *Contribution* relations are used when the relation between goals is not the consequence of a deliberative planning but rather results from side-effects. Therefore, contribution relations propagate neither objectives, capabilities, nor entitlements. The impact can be positive or negative and is graphically represented as edges labeled with "+" and "−", respectively. Finally, tasks are linked to the goals that they intend to achieve using *means-end* relations.

The relations between actors within the system are captured by the notions of delegation and trust. Assignment of responsibilities among actors can be made by *execution dependency* (when an actor depends on another actor for the achievement of a goal) or *permission delegation* (when an actor authorizes another actor to achieve the goal). Usually, an actor prefers to appoint actors that are expected to achieve assigned duties and not misuse granted permissions. SI* adopts the notions of *trust of execution* and *trust of permission* to model such expectations. In the graphical representation, permission delegations are represented with edges labeled by **Dp** and execution dependencies with edges labeled by **De**. Finally, trust of permission relations are represented with edges labeled by **Tp** and trust of execution relations with edges labeled by **Te**.

To illustrate what SI* models are, let us look at a health care scenario (Figure 1). This example is an excerpt of a case study analyzed in the EU SERENITY project[3] and we will use it through the paper to demonstrate our approach.

Example 1. Patients depend on the Health Care Centre (HCC) for receiving medical services, such as assistance because of faintness alert and home delivery of medicines. When a patient feels giddy, he can send a request for assistance to the Monitoring and Emergency Response Centre (MERC), a department of the HCC. The MERC starts a doctor discovery process that consists in sending a message to a group of doctors. The first doctor that answers the request is appointed to provide medical care to the patient. The selected doctor sets a diagnosis and defines the necessary treatments in terms of medical prescriptions or requests for specialist visits. A patient can also require the MERC to get medicines from the pharmacy. In this case, a social worker is contacted by the MERC to go to the pharmacy and get the medicine to be delivered to the patient.

[3] http://www.serenity-project.org

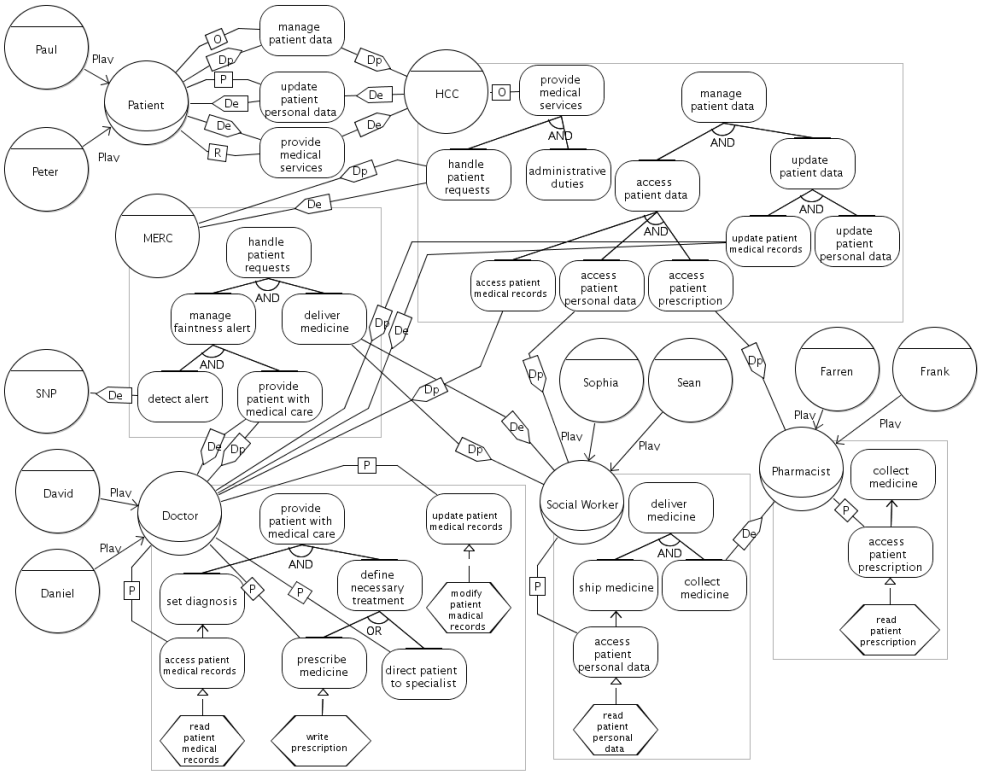

Fig. 1. A SI* model for the health care scenario

3 Access Control Policy Specification

Access control policies shall be compliant with organizational and system requirements. The key idea to the modeling of access rights to data objects is that actors have some kind of permission with respect to the activities they have to perform. Here, we propose a methodological framework for supporting policy writers in the specification of access control policies from the functional and security requirements of the system. The framework intends to analyze the organizational context in terms of the actors who comprise it, their goals (i.e., entitlements, objectives, and capabilities), and the interrelations among them. Goals are then operationalized into specifications of operations to achieve them. The access control policy is defined as the set of permission associated with operations. In the remainder of this section, the phases of the access control specification process (Figure 2) are presented in detail.

Actor modeling (step 1) aims to identify and model the roles (e.g., Doctor, Social Worker, etc.) and agents (e.g., HCC, MERC, David, etc.) within the socio-technical system. Identified actors are described along with their objectives, entitlements, and capabilities. In this phase, agents are also described along the

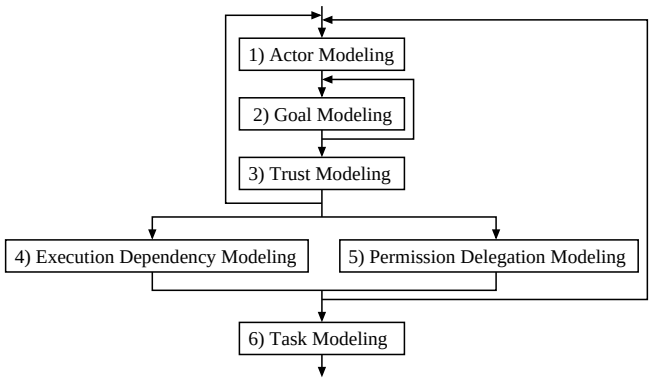

Fig. 2. Access Control Policy Specification Process with SI*

roles they play (e.g., David *plays* role Doctor). As the analysis proceeds (steps 3, 4, and 5), more actors might be identified, leading to new iterations of the actor modeling phase.

Goal modeling (step 2) aims to analyze objectives, entitlements and capabilities of actors. These are analyzed from the perspective of their respective actor using the various forms of goal analysis described earlier. Initially, goal modeling is conducted for all root goals associated with actors. Later on, more goals are created as goals are delegated to existing or new actors (steps 4 and 5). The refinement of goals are considered to reach an adequate level once objectives of every actor have been assigned to actors that are capable to achieve them and permission are specified at an appropriate level of granularity.

The assignment of responsibilities is driven by the expected behavior of other actors. *Trust modeling* (step 3) enriches the requirements model by identifying trust relationships (of both permission and execution) between actors. *Execution dependency modeling* (step 4) aims to discover and establish dependencies between actors (e.g., the Patient *depends* on the HCC for achieving goal provide medical services). Entitlements are also analyzed from the perspective of each actor. *Permission delegation modeling* (step 5) aims to identify and model transfers of authority between actors (e.g., the Patient *delegates* the *permission* to manage patient data to the HCC).

Once all objectives have been dealt with to the satisfaction[4] of the actors who want them, *task modeling* (step 6) identifies the actions to be executed in order to achieve goals. For the sake of simplicity, in this work we assume that tasks are atomic actions (e.g., read, write, modify, etc.) and cannot be further refined. The identified tasks are linked to goals via means-end relations that propagate properties of goals (i.e., objectives, entitlements, and capabilities) to tasks. The access control policy is defined as the set of permissions associated with tasks.

[4] An actor A can satisfy a goal G if G is an objectives of A and A has the capabilities to achieve G or A depends on an actor who can satisfy G [18].

(Doctor,read patient medical data)
(Doctor,modify patient medical data)
(Doctor,write prescription)
(Social Worker,read patient personal data)
(Pharmacist,read patient prescription)

Fig. 3. Access Control Policy

Example 2. Figure 3 shows the access control policy derived from the health care scenario of Figure 1. The Doctor is appointed by the HCC to provide patient with medical care and update patient medical records. The requirements analysis process shows that the Doctor shall be authorized to read and modify patient medical records as well as write prescription to achieve assigned duties. Similarly, the Social Worker shall be authorized to read patient personal data for shipping medicine and the Pharmacist shall be authorized to read patient prescription for collecting medicine. It is worth noting that the Social Worker has not the permission to read the prescriptions; only the Pharmacist is authorized to do it. As consequences, the system designer needs to employ some mechanism to protect prescriptions, for instance, by enclosing them into closed and sealed envelopes that only the pharmacist is authorized to open[5] (if prescriptions are in paper form) or by encrypting them (if prescriptions are in digital form).

In the above example we showed a RBAC policy as we have only considered the access rights to be associated with roles. The framework is, however, flexible enough to specify policies in other access control models. This might be useful, for instance, when access rights are specified with respect to agents instead of to roles.

4 Policy Verification at Organizational Level

The most frequent question during modeling is whether the policy is consistent with functional requirements and compliant with security requirements. The first issue we tackle concerns the analysis of functional requirements from the perspective of actors who want goals achieved (hereafter requesters). We verify whether actors' objectives are satisfied by the system design.

Pro1 Every requester has assigned (possibly indirectly) the achievement of his objectives to actors that have the capabilities and the necessary access right to achieve them.

Pro2 Every requester has assigned (possibly indirectly) the achievement of his objectives to actors that he trusts.

The first property verifies that actors' objectives are assigned to actors that can actually take charge of their satisfaction. The latter aims to provide additional

[5] The pharmacist can report the occurrence of a misuse to the HCC if he receives an envelope without the seal or with a broken seal from the social worker.

guarantee about the satisfaction of goals by employing the notion of trust. The satisfaction of these properties ensures requirements engineers that the system design leads to the satisfaction of the objectives of each actor.

From the perspective of actors who control the achievement of goals (hereafter owners), the system design should guarantee that entitlements of actors are not misused. To this purpose, we employ the following properties:

Pro3 Entitlements have been assigned only to actors trusted by their owners.
Pro4 Actors granting permissions to achieve a goal, have the right to do so.
Pro5 Entitlements have been assigned to actors who actually need them to perform their duties.

Pro3 provides owners with assurance that their entitlements are used properly. The system design has also to ensure that actors do not grant privileges they do not have to other actors. This is verified by Pro4. Pro5 verifies the compliance of the model with the least privilege principle.

Finally, designers have to analyze their model from the perspective of actors that are actually in charge of achieving goals (hereafter providers).

Pro6 Every provider has the permissions necessary to accomplish assigned duties.

A failure of Pro6 can be due to the lack of assignments of permission to legitimate users. In this case, designers should revise the model by identifying the permission path that at the end will authorize the provider to achieve the goal. The failure, however, can be also due to the fact that the achievement of objectives has been assigned to actors that should not be authorized to accomplish such tasks. This requires designers to revise the model by identifying other actors that has the capabilities to achieve the goals. It is worth noting that only by modeling security and functional requirements separately one is able to capture both these aspects. Indeed, by deriving access control policies from the functional requirements one always ends up that access rights have not been assigned to users.

5 Policy Verification at User Level

The analysis of industry case studies (e.g., [31, 40]) has revealed that security breaches are often not apparent in policies specified at organizational level. They only appear at the instance level, once we see what happens when individuals are mapped into roles and delegation paths are concretely followed. Requirements engineers, however, do not usually want to design the system at the instance level even if they need to reason at that level to detect many security breaches. Our objectives is thus to offer them tools for instantiating organizational requirements and analyzing the instantiated requirements.

The SI* modeling framework allows for a clear distinction between organizational and instance levels as it only employs the notions of agent and role.

The organizational level focuses on roles by associating with each role the objectives, entitlements, and capabilities related to the activities that such a role has to perform within the organization and the relationships among them. The instance level focuses on single agents by identifying their personal objectives, entitlements, and capabilities and the relationships among them as well as the roles they play.

Many access control frameworks determine the permissions assigned to users (possibly via roles assignment) by adopting an inheritance method, that is, a user inherits all privileges associated with the roles he plays. This approach, however, requires access rights to be specified on concrete objects, making difficult policy specification, analysis, and management. In this section, we propose an instantiation procedure that automatically relates goal instances to users with respect to the responsibilities and permissions that have been assigned to them. The basic idea is that when a designer draws a goal, he specifies an "abstract goal", rather than a "goal instance". Then, it is matter of the instantiation procedure to automatically generate the instantiated requirements model.

Different SI* concepts instantiate goals and social relations differently. Though it might seem that the instantiation of objectives depends on the responsibilities of agents, an agent will unlikely desire the fulfillment of all instances of a goal. Rather, he is interested in the achievement of a particular instance. For example, a patient (e.g., Peter in Figure 2) cares about medical services provided to him rather than to services provided to other patients. There might be situations where agents want to satisfy more than one instance of the same goal. This is, for instance, the case of the HCC that is in charge of providing medical treatments to those agents, who requested them. However, they are delegated responsibilities. Conversely, an agent to whom capabilities are prescribed (possibly via role assignment), has the capabilities to achieve all instances of a goal. For instance, a doctor (e.g., David) is capable of prescribing medicines to all patients rather than only to one particular patient.

The instantiation of entitlements is on case-by-case basis. The designer may assign permission to a role with the intended meaning that the agents that play that role are entitled to control only a particular instance of the goal. On the contrary, there are situations where an actor is entitled to control all instances of the goal. The difference in the meaning of entitlement is evident by looking at the relationship between the patient and his data and between the HCC and medical services (Figure 2). When a designer says that "a patient is entitled to control the use of patient's medical records", he means that every patient is entitled to control only his own medical records. Conversely, the HCC has full authority over all instances of the provisioning of medical services. To distinguish these situations, we have refined the concept of ownership into existential ownership and universal ownership. *Existential ownership* indicates that the actor is the legitimate owner of one instance of the goal. *Universal ownership* indicates that the actor is the legitimate owner of all instances of the goal.

Execution dependencies (permission delegations, resp.) propagate the responsibility (authority, resp.) to achieve the goal instances generated by objective (ex-

istential ownership, resp.) instantiation to the agents playing the role of dependee (delegatee, resp.). However, only one instance of the dependee is appointed to perform the assigned duties. This intuition is actually closer to reality than one may think: when the MERC appoints a doctor to provide medical care to a patient, only one doctor will perform this task. Similarly, permission delegations grant permission only to one of the agents playing the role of delegatee. Trust relations are used to model the expectation of an actor. This expectation is not related to a particular goal instance but refers to the general behavior of the trustee, that is, to all instances of the goal. Accordingly, trust relations are instantiated for all instances of the goal. Moreover, trust relations are instantiated between every agent playing the role of trustor and every agent playing the role of trustee.

This instantiation model can also assist system and security administrators in the configuration decision making process. For instance, the simple SI* model in Figure 2 generates a huge number of possible configurations (i.e., combinations of assignments of objectives and entitlements).[6] However, not all of them may guarantee a fair and lawful behavior of the socio-technical system. For instance, in many configurations access rights are assigned to agents that are not actually in charge to achieve the goals. The key idea is to apply the properties presented in Section 4 to the instantiated model in order to discard those configurations that are not compliant with security and organizational requirements.

6 Domain-Specific Verification

The analysis at the instance level allows us to capture other situations that might result harmful to the system or one of the stakeholders of the system.

Example 3. The first doctor who answer Peter's request is David. Thereby, David is appointed by the MERC to provide medical care to Peter. David is also a consultant in the health insurance company with whom Peter has stipulated an insurance policy. This situation is clearly to be avoided. The MERC needs to find another doctor to provide medical care to Peter. This demands a revision of the doctor discovery procedure: the first-answer first-appointed policy (see Example 1) should be modified by introducing a check for possible conflicts.

If we look at ways to handle situations like the above example, starting from the landmark paper by Saltzer and Schroeder [37] to other classical papers [1, 15, 16, 19, 41], we found that Separation of Duty (SoD) is invariably offered as "the" solution to prevent the violation of business rules. SoD aims to reduce the risk of security breaches by not allowing any individual to have sufficient authority within the system to compromise it on his own [7]. Our framework supports the specification of SoD constraints at three levels of granularity. The basic type of constraint simply denies agents to play conflicting roles.

[6] The number of configuration is exponential in the number of OR-decompositions, permission delegations, execution dependencies, and agent-role assignments.

Example 4. A doctor in the HCC shall not work as a consultant in an insurance company.

A second type focuses on incompatible activities. They prevent users from performing activities whose combination can compromise the system integrity.

Example 5. A doctor shall not provide both health care in behalf of the HCC and consulting to the insurance company. This constraint, however, does not deny doctors to perform other duties within the HCC and the insurance company.

Above types of constraint can be classified as static SoD constraints [41]. In some cases they impose too strict limits on requirements. To this end, the framework allows for the specification of dynamic SoD constraints [41] by focusing on particular instances of activities.

Example 6. David shall not provide assistance to patients who have stipulated an insurance policy with the insurance company were he works. This constraint, however, does not deny David to provide medical care to patients who do not have any relation with the insurance company.

Other constraints imposed by the application domain can be defined, for instance, to specify the cardinality of roles, that is, the number of agents that can play a role at the same time, or the number of tasks assigned to single agents.

7 Automated Reasoning Support

Looking at the process in Figure 2, it is evident the need of tools to assist policy writers in determining (1) which actors' goals are satisfied and (2) the permission on tasks. These activities can be cumbersome to be manually performed especially when the requirements model is huge. Tool support is also necessary for model instantiation and policy verification.

For our purpose, we have chosen the ASP paradigm with value invention [9]. In [18] the authors have defined the semantics of SI* in the ASP paradigm [26]. Roughly speaking, ASP is a variant of Datalog with negation as failure and disjunction. This paradigm supports specifications expressed in terms of facts and Horn clauses, which are evaluated using the stable model semantics. Here, graphical models are encoded as sets of facts (see [29] for details on the trasformation of graphical models into formal specifications). Rules (or axioms) are Horn clauses that define the semantics of SI* concepts. Specifically, axioms are used to propagate objectives, entitlements, and capabilities across the requirements model via goal analysis, execution dependencies, and permission delegations. As an example, we report the axioms for entitlements and permission delegations (Ax1-3) in Table 1 [18]. As described earlier in the paper, the access control policy is defined as the set of permission associated to tasks. Ax4 is used to determine such permission. Axioms are also used to determine the goals that can be satisfied and to propagate satisfaction evidence backward to the requester [18].

Ax1 $\mathsf{have\_perm}(X, G) \leftarrow \mathsf{own}(X, G)$
Ax2 $\mathsf{have\_perm}(X, G) \leftarrow \mathsf{delegate}(Y, X, G) \wedge \mathsf{have\_perm}(Y, G)$
Ax3 $\mathsf{have\_perm}(X, G_1) \leftarrow \mathsf{subgoal}(G_1, G) \wedge \mathsf{have\_perm}(X, G)$
Ax4 $\mathsf{access\_control}(X, T) \leftarrow \mathsf{means\_end}(T, G) \wedge \mathsf{have\_perm}(X, G)$

Table 1. Axiomatization of Entitlements and Permission Delegations

Pro3 $\leftarrow \mathsf{own}(X, G) \wedge \mathsf{not\ confident\_owner}(X, G)$
Pro4 $\leftarrow \mathsf{delegate}(X, Y, G) \wedge \mathsf{not\ have\_perm}(X, G)$
Pro5 $\leftarrow \mathsf{have\_perm}(X, G) \wedge \mathsf{not\ need\_to\_have\_perm}(X, G)$
Pro6 $\leftarrow \mathsf{need\_to\_have\_perm}(X, G) \wedge \mathsf{not\ have\_perm}(X, G)$
SoD $\leftarrow \mathsf{play}(A, r_1) \wedge \mathsf{play}(A, r_2)$
RC $\leftarrow \#\mathsf{count}\{X : \mathsf{play}(A, r)\} > n$

Table 2. Properties of Design and Domain-Specific Constraints

Properties of design (Section 4) and domain-specific constraints (Section 6) are encoded as ASP constraints. Constraints are Horn clauses without positive literals and are used to specify conditions which must not be true in the model. In other words, constraints are formulations of possible inconsistencies. Table 2 presents some examples of constraints. Pro3 verifies if an owner is confident that there is no likely misuse of his entitlements. Specifically, an owner is confident that permissions on his entitlements have been assigned only to trusted actors. Here, literal $\mathsf{confident\_owner}(x, g)$ holds if actor x is confident that permissions on goal g are given only to trusted actors. Pro4 verifies that actors, who delegate the permission to achieve a goal, are entitled to do it, that is, it that checks that permission are well rooted. Pro5 verifies that actors, who have the permission to achieve a goal, actually need such permission. Pro6 is opposite to Pro5. It verify that actors, who need to have the permission to achieve their duties, have such permission. Thereby, the combination of Pro5 and Pro6 guarantees that actors have access right if and only if they need them. SoD verifies that there are no agents that play both roles r_1 and r_2. RC verifies that there are not more than n agents that play role r.[7]

Facts, axioms and constraints compound the program that is executed by an ASP inference engine. As result, the engine returns all answer sets (i.e., sets of atoms) satisfying all Horn clauses. These answer sets represent the system configurations in which all properties of design are satisfied. Answer sets include the access control policy, that is, the sets of facts in the form $\mathsf{access\_control}(x, t)$.

The ASP paradigm, however, is not sufficient for implementing the instantiation procedure presented in Section 5. ASP with value invention improves ASP by introducing function symbols. Essentially, functions are treated as external predicates that implement the mechanism of value invention by taking in input a set of values and returning a new value. Accordingly, instances of goals are represented using function $g_i(g, a, r)$, where g is a goal, a is the agent who has generated the instance, and r is the role from which the agent has taken

[7] $\#\mathsf{count}\{\ldots\}$ is a built-in aggregate function that is supported by several ASP solvers.

$$\text{I1 } \mathsf{own_i}(A, g_i(G, A, R)) \leftarrow \mathsf{own\_existential}(R, G) \wedge \mathsf{play}(A, R)$$

$$\text{I2 } \mathsf{own_i}(A, g_i(G, B, R)) \leftarrow \begin{cases} \mathsf{own\_universal}(P, G) \wedge \mathsf{play}(A, P) \wedge \\ \mathsf{agent}(B) \wedge \mathsf{role}(R) \wedge \mathsf{goal}(G) \end{cases}$$

$$\text{I3 } \mathsf{delegate_i}(A, B, g_i(G, C, R)) \leftarrow \begin{cases} \mathsf{delegate}(P, Q, G) \wedge \mathsf{play}(A, P) \wedge \mathsf{play}(B, Q) \wedge \\ \mathsf{have\_perm}(A, g_i(G, C, R)) \wedge \\ \mathsf{not\ other\_agent}(B, Q, g_i(G, C, R)) \end{cases}$$

$$\text{I4 } \mathsf{other\_agent}(A, R, G) \leftarrow \mathsf{play}(A, R) \wedge \mathsf{play}(B, R) \wedge \mathsf{delegate_i}(D, B, G) \wedge A \neq B$$

Table 3. Instantiation of Entitlements and Permission Delegations

the goal.[8] The choice of implementing the instantiation procedure in ASP with value invention instead of in other formalisms allows us to reuse the framework proposed in [18] (with some minor changes). Actually, the rules for instantiation are simply added to the ASP program used for the analysis of organizational requirements. Table 3 presents the rules for instantiating entitlements and permission delegations.[9]

One can observe the different instantiation for existential and universal ownership. Specifically, the rule for existential ownership (I1) introduces new values, that is, it creates new instances of the goal. On the other hand, the rule for universal ownership (I2) considers all the instances of the goal. Rule I3 implements the instantiation of permission delegations. We introduce predicate other_agent to verify if the permission has already been assigned to another agent. Thus, axiom I3 (in combination with I4) will not yield one model but multiple models in which the permission is granted only to one agent playing the role of the delegatee. Another observation concerns the goal instance: an actor can delegate only the permission on the instances that are in his scope, that is, instances which the actor is already entitled to achieve. The proposed approach has been implemented in the DLV system [26] – a state-of-the art implementation of ASP.

8 Related Work

Several languages and models intended to support policy writers and system administrators in the specification and enforcement of access control policies has been proposed [6, 23, 33, 39]. Our work is complementary to those proposals. Indeed, we have not proposed a new access control language. Rather, our objective is to support policy writers in defining access control policies, which can be specified using existing languages.

Several efforts have been spent to close the gap between security requirements analysis and policy specification. Basin et al. [4] propose SecureUML, an UML-based modeling language for modeling access control policies and integrating them into a model-driven software development process. Similar approaches have been proposed by Doan et al. [14], who incorporate Mandatory Access Control

[8] We use the constant null when the goal is directly associated to an agent.

[9] For the sake of simplicity, Table 3 reports the rules used when own and delegation are specified for roles. Similar rules are used when relations are specified for agents.

(MAC) into UML, and by Ray et al. [35], who model RBAC as a pattern using UML diagram template. Breu et al. [8] propose an approach for the specification of user rights in the context of an object oriented use case driven development process. However, these frameworks do not provide facilities for the analysis of the social context where the system operates.

The problem of specifying access control policies has been partially addressed in workflow management systems. For instance, Bertino at al. [7] formally express constraints on the assignment of roles to tasks in a workflow in order to automatically assign users to roles according to such constrains. Kang at al. [25] propose a fine-grained and context-based access control mechanism for inter-organizational workflows. The idea underlying these proposals is to grant access rights to users on the basis of the duties assigned to the roles they play. This approach, however, does not allow the analysis of the functional requirements of the system to be protected.

Moving towards early requirements, He et al. [20] propose a goal-driven framework for modeling RBAC policies based on role engineering [10]. This framework includes a context-based data model, in which policy elements are represented as attributes of roles, permissions, and objects, and a goal-driven role engineering process, which addresses how the security contexts in the data model can be elicit and modeled. However, roles are derived from task and are not analyzed with respect to the organizational context. Liu et al. [28] propose an access control analysis in i*. The main difference with our approach lies in the degree of automation. Liu et al. provide a systematic way to specify access control policies, but leave all work to humans. Indeed, they do not provide any tool support for assisting policy writers in their work. Moreover, similarly to workflow access control proposals, the authors propose to grant a permission to an actor every time he needs such a permission. Crook et al. [11] enhance i* to derive role definition from the organizational context. However, instantiation is still manually. Moreover, as in [7, 25] permissions are simple derived by the tasks assigned to users, leaving a little room for verifying the consistency between security and functional requirements. Finally, we mention the work by Fontaine [17], who propose a mapping of goal models based on KAOS [13], a goal-based requirements engineering methodology, onto Ponder [12], a language for specifying management and security policies for distributed systems. The key point of this work is the transformation of operationalized goals into access control policies. However, KAOS is inadequate to model and analyze policies because it lacks the necessary features necessary for the modeling and analysis of the organization structure.

In the area of policy verification, Sohr et al. [42] propose a framework for the verification and validation of RBAC policies and authorization constraints. Policies and constraints are specified as sentences in first-order LTL and verified using theorem provers. Although the use of theorem provers allows analysts to give proofs that are independent from the number of users and objects, it makes the verification process not completely automated. The authors also propose a validation approach based on UML and OCL: RBAC policies are modeled as

class diagrams and authorization constraints are specified in OCL. The UML-based Specification Framework is then used to generate system states and to check those states against specified constraints. Hu et al. [22] propose a framework for verification and conformance testing for secure system development. Verification is intended to ensure that access control policies comply with security properties, and conformance testing is used to validate the compliance of system implementation with access control policies. However, these proposals mainly focus on the implementation and enforcement of access control policies and do not provide any methodological support for the analysis of the organizational context and the definition of access control policies on the basis of elicited organizational and system requirements.

9 Conclusive Remarks and Future Work

In this paper we have proposed a model driven approach to assist policy writers in the specification and analysis of access control policies with respect to organization and security requirements and system administrators in the user-permission assignment decision making. To support a more accurate analysis, we have defined an approach for instantiating organizational requirements. Readers familiar with RBAC and other access control models will easily find out some differences in the way permissions are assigned to agents. In RBAC a user inherits all permissions associated with the roles he plays. If permission is specified for classes of objects, the user is entitled to access all objects in those classes [23]. We observe that this assumption is not always true especially with regards to the least privilege principle. For example, just because the doctor role can access a patient record does not mean that a doctor can access all patient records. A doctor can only access the records of those patients currently assigned to that doctor.

The instantiation procedure together with policy analysis facilities have been implemented in ASP with value invention. One may claim that the approach suffers from exponential complexity. We argue that this is *the cost of security*: policy writers shall explore the entire space of solutions to identify vulnerabilities in their access control policies. Scalability problems, however, occur at design time where the policy writer can add and remove agents for a more accurate analysis. They disappear at run time when administrators verify whether or not the actual system configuration is secure. Indeed, the problem of verifying if a certain configuration satisfies properties of design and domain-specific constraints is polynomial. The last observation concerns the verification of Pro4 against the instantiated requirements. Rule I3 instantiates permission delegations only for the instances in the scope of the agent. This approach makes every permission well rooted by construction. The verification of Pro4 at the instance level, however, can be done by creating a "fake" instance of the goal, for instance, when the agent is supposed to delegate the permission on a goal but he has not it on any instance of that goal.

The research presented here is still in progress. Much remains to be done to further refine the proposed approach to support the specification of access control policies comparable to the ones that can be expressed, for instance, in XACML [33]. Future work plans include the support for the specification of negative authorizations and obligations. Another direction under investigation involves the capture of behavioral aspects by means of revocation policies.

References

1. G.-J. Ahn and R. Sandhu. The RSL99 language for role-based separation of duty constraints. In *Proc. of RBAC'99*, pages 43–54. ACM Press, 1999.
2. A. I. Antón and C. Potts. The use of goals to surface requirements for evolving systems. In *Proc. of ICSE'98*, pages 157–166. IEEE Press, 1998.
3. Association of Certified Fraud Examiners. The 2006 report to the nation, 2006.
4. D. Basin, J. Doser, and T. Lodderstedt. Model Driven Security: from UML Models to Access Control Infrastructures. *TOSEM*, 15(1):39–91, 2006.
5. M. Y. Becker and P. Sewell. Cassandra: flexible trust management, applied to electronic health records. In *Proc. of CSFW'04*, pages 139–154. IEEE Press, 2004.
6. D. E. Bell and L. J. LaPadula. Secure Computer System: Unified Exposition and MULTICS Interpretation. Technical Report MTR-2997 Rev. 1, The MITRE Corporation, Bedford, MA, 1976.
7. E. Bertino, E. Ferrari, and V. Atluri. The specification and enforcement of authorization constraints in workflow management systems. *TISSEC*, 2(1):65–104, 1999.
8. R. Breu, G. Popp, and M. Alam. Model based development of access policies. *STTT*, 9:457–470, 2007.
9. F. Calimeri and G. Ianni. External Sources of Computation for Answer Set Solvers. In *Proc. of LPNMR'05*, LNCS 3662, pages 105–118. Springer-Verlag, 2005.
10. E. J. Coyne. Role engineering. In *Proc. of RBAC'95*, pages 15–16. ACM Press, 1995.
11. R. Crook, D. Ince, and B. Nuseibeh. On Modelling Access Policies: Relating Roles to their Organisational Context. In *Proc. of RE'05*, pages 157–166, 2005.
12. N. Damianou, N. Dulay, E. Lupu, and M. Sloman. The Ponder Policy Specification Language. In *Proc. of POLICY'01*, LNCS 1995, pages 18–39. Springer-Verlag, 2001.
13. A. Dardenne, A. van Lamsweerde, and S. Fickas. Goal-directed Requirements Acquisition. *Sci. of Comp. Prog.*, 20:3–50, 1993.
14. T. Doan, S. Demurjian, T. C. Ting, and A. Ketterl. MAC and UML for secure software design. In *Proc. of FMSE'04*, pages 75–85. ACM Press, 2004.
15. J. E. Dobson and J. A. McDermid. A framework for expressing models of security policy. In *Proc. of Symp. on Sec. and Privacy*, pages 229–239. IEEE Press, 1989.
16. D. F. Ferraiolo, J. F. Barkley, and D. R. Kuhn. A role-based access control model and reference implementation within a corporate intranet. *TISSEC*, 2(1):34–64, 1999.
17. P.-J. Fontaine. *Goal-Oriented Elaboration of Security Requirements.* PhD thesis, Université Catholique de Louvain, 2001.
18. P. Giorgini, F. Massacci, and N. Zannone. Security and Trust Requirements Engineering. In *FOSAD 2004/2005*, LNCS 3655, pages 237–272. Springer-Verlag, 2005.

19. V. D. Gligor, S. I. Gavrila, and D. Ferraiolo. On the formal definition of separation-of-duty policies and their composition. In *Proc. of Symp. on Sec. and Privacy*, pages 172–183. IEEE Press, 1998.
20. Q. He and A. I. Antón. A Framework for Modeling Privacy Requirements in Role Engineering. In *Proc. of REFSQ'03*, pages 137–146, 2003.
21. House of Lords. Prince Jefri Bolkiah vs KPMG. 1 All ER 517, 1999.
22. H. Hu and G. Ahn. Enabling verification and conformance testing for access control model. In *Proc. of SACMAT'08*, pages 195–204. ACM Press, 2008.
23. S. Jajodia, P. Samarati, M. L. Sapino, and V. S. Subrahmanian. Flexible support for multiple access control policies. *TODS*, 26(2):214–260, 2001.
24. J. Jürjens. *Secure Systems Development with UML*. Springer-Verlag, 2004.
25. M. H. Kang, J. S. Park, and J. N. Froscher. Access control mechanisms for inter-organizational workflow. In *Proc. of SACMAT'01*, pages 66–74. ACM Press, 2001.
26. 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.
27. N. Li and J. C. Mitchell. RT: A Role-based Trust-management Framework. In *Proc. of DISCEX'03*, volume 1, pages 201–212. IEEE Press, 2003.
28. 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.
29. F. Massacci, J. Mylopoulos, and N. Zannone. Computer-Aided Support for Secure Tropos. *ASE*, 14(3):341–364, 2007.
30. F. Massacci, J. Mylopoulos, and N. Zannone. An Ontology for Secure Socio-Technical Systems. In *Handbook of Ontologies for Business Interaction*, chapter XI, page 188. The IDEA Group, 2008.
31. 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, 2008. To appear.
32. D. Mellado, E. Fernández-Medina, and M. Piattini. Applying a Security Requirements Engineering Process. In *Proc. of ESORICS'06*, volume 4189 of *LNCS*, pages 192–206. Springer-Verlag, 2006.
33. OASIS. eXtensible Access Control Markup Language (XACML) Version 2.0. OASIS Standard, 2005.
34. Promontory Financial Group, Wachtell, Lipton, Rosen, and Katz. Report to the Board and Directors of Allied Irish Bank P.L.C., Allfirst Financial Inc., and Allfirst Bank Concerning Currency Trading Losses, March 12, 2003.
35. 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.
36. S. Room. *Data Protection & Compliance in Context*. BCS, 2007.
37. J. H. Saltzer and M. D. Schroeder. The Protection of Information in Computer Systems. *Proceedings of the IEEE*, 63(9):1278–1308, 1975.
38. P. Samarati and S. D. C. di Vimercati. Access Control: Policies, Models, and Mechanisms. In *FOSAD 2001/2002*, LNCS 2946, pages 137–196. Springer-Verlag, 2001.
39. R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-Based Access Control Models. *IEEE Comp.*, 29(2):38–47, 1996.
40. A. Schaad, V. Lotz, and K. Sohr. A model-checking approach to analysing organisational controls in a loan origination process. In *Proc. of SACMAT'06*, pages 139–149. ACM Press, 2006.
41. R. Simon and M. E. Zurko. Separation of duty in role-based environments. In *Proc. of CSFW'97*, pages 183–194. IEEE Press, 1997.

42. K. Sohr, M. Drouineaud, G.-J. Ahn, and M. Gogolla. Analyzing and managing role-based access control policies. *TKDE*, 20(7):924–939, 2008.