Conflict Detection in Security Policies using Semantic Web Technology

Mario Arrigoni Neri, Marco Guarnieri, Eros Magri, Simone Mutti, Stefano Paraboschi
Dipartimento di Ingegneria dell'Informazione e Metodi Matematici,
Università degli Studi di Bergamo, Italy
Email: {mario.arrigonineri, marco.guarnieri, eros.magri, simone.mutti, parabosc}@unibg.it

***Abstract*—The design of efficient and effective techniques for security policy analysis is a crucial open problem in modern information systems. Significant attention has been dedicated in the past to the use of logical tools for the analysis of security policies, but this work has produced a limited impact on enterprise information systems. Important reasons of the limited success of past research were the difficult integration of these approaches with the technological scenario and the limited scalability of many proposals.**

Nowadays Semantic Web tools are increasingly used in modern information systems. We show how the tools provided by Semantic Web and ontology management technologies offer an adequate basis for the realization of techniques able to support conflict analysis in security policies. Based on the use of these techniques, we propose a solution for two different variants of conflict analysis: (a) *Policy Incompatibility*, and (b) *Separation of Duty Satisfiability*. Experiments that test the techniques on large security policies demonstrate the scalability of the approach.

I. Introduction

The evolution of information systems sees a continuously increasing need of flexible and sophisticated approaches for the management of security requirements. On one hand, systems are increasingly more integrated and present interfaces for the invocation of services accessible through network connections. On the other hand, system administrators have the responsibility to guarantee that this integration and the consequent exposure of internal resources does not introduce vulnerabilities. The need to prove that the system correctly manages the security requirements is not only motivated by the increased exposure, but also by the need to show compliance with respect to the many regulations promulgated by governments and commercial bodies. The integrated management of security policies promises to produce significant benefits in this area in terms of better and more efficient protection.

In a way similar to the evolution seen in the area of software development and database design, the model-driven engineering of security leads to the specification of security requirements at an abstract level, with the subsequent refinement of the abstract model toward a concrete implementation. It is then guaranteed that, when the high-level representation of the security requirements is correct, the security configuration of the system satisfies the requirements. The application of this approach requires the implementation of a rich collection of tools supporting, for each of the many components in the system, the refinement from the high-level representation of security requirements to the concrete security configuration. A significant investment in research and development is needed to realize this vision, in order to manage the heterogeneous collection of devices and security services that information and communication technology offers to system administrators. Still, this large investment promises to produce adequate returns, considering the importance of the correct management of security requirements in modern information systems.

A crucial advantage of the model-driven approach described above is the possibility of an early identification of anomalies in the security policy. Security policies in real systems often exhibit conflicts, i.e., inconsistencies in the policy that can lead to an incorrect realization of the security requirements. The availability of a high-level and complete representation of the security policies supports the construction of services for the analysis of the policies able to identify these anomalies and suggest corrections.

Although security administrators can in many cases manually handle simple access control policies (ACLs), manual approaches cannot scale well to bigger policies. For instance, maintaining and modifying complex ACLs is not an easy task, because it may be difficult to identify all the possible side effects of a change in the policy. Although some current access control languages (e.g., XACML) implement techniques that automatically solve conflicts between rules in the policy by means of run-time conflict resolution strategies, anomaly detection at design-time provides a valuable help to security administrators, because it is difficult for them to clearly understand what the results of the run-time conflict resolution algorithm will be, at least in complex policies, and this can lead to unexpected problems and misconfigurations. Due to these needs, the identification of conflicts in policies has been the subject of a significant amount of research in the last years. Specifically, Semantic Web and ontology management technology [9], [10] today offers a variety of interesting solutions. In the recent past, the application of logical models to the analysis of security policies required to use logical tools that had no clear integration with common information system tools; in most cases administrators were unfamiliar to them and they perceived the risk of limited scalability [14]. Modern Semantic Web tools are instead well integrated with existing XML and Web technology, are used

The research leading to these results has received funding from the European Commission Seventh Framework Programme (FP7/2007-2013) under grant agreement no. FP7-ICT-257129 "PoSecCo".

in many scenarios to efficiently obtain solutions for several optimization problems, and for this reason are often already familiar to administrators. The familiarity of computer science practitioners with Semantic Web tools is likely going to increase, as the diffusion of these tools increases.

This paper will present experimental results that will confirm that the use of Semantic Web tools for policy analysis permits the adoption of a variety of approaches, with a choice that depends on the complexity of the analyzed property and the size of the policy. A *one-size-fits-all* strategy cannot be considered adequate.

Section II presents the base model. Section III illustrates the *Policy Incompatibility* service. Section IV is dedicated to the analysis of the *Separation Of Duty Satisfiability* service. Section V reports on experimental results, using as representative policies a model built over bibliographical databases. In Section VI a comparison is made with previous related work. Section VII draws a few concluding remarks.

II. Base model

In enterprise scenarios, policy definition and management systems are needed that provide a high level of flexibility, to correctly represent the large number and variety of security requirements. A contribution to this accomplishment is given by the use of rich metamodels for involved entities, as well as of a rich language to describe relationships between the entities. Support for the representation of rich domain entities can be obtained by applying techniques from the *Semantic Web* and knowledge representation area, which offer expressivity and the possibility to specify over the model flexible constraints and derived information.

Our model presents four first-level concepts:

- **Principal**: it represents an *Identity* or a *Role*. An *Identity* can be a *SingleId*, which represents a single user, or a *Group*, which represents a group of users. *Roles* represent the binding between principals and the set of permissions associated with a specific enterprise function. The *Principals* hierarchy is expressed by the following DL formulas, $Identity \sqsubseteq Principal$, $Role \sqsubseteq Principal$, $SingleIdentity \sqsubseteq Identity$, $Group \sqsubseteq Identity$ and $SingleIdentity \sqcap Group \sqsubseteq \perp$. Identities are organized in groups by the property[1] *containedIn*: *Identity* $\rightarrow$ *Group*[2] and its inverse *contains* = *containedIn*$^{-1}$. A *Group* can contain *SingleIdentities* and other *Groups*. The $roleHierarchy$: *Role* $\rightarrow$ *Role* property connects each role to its direct sub-roles. Its transitive closure *canBe+*: *Role* $\rightarrow$ *Role* can be used to identify all the direct or indirect sub-roles.
- **Resource**: it represents the elements (data and services) in the information system, whose access must be regulated through security policies. For the containment hierarchy of resources, we introduce the property *recourceHierarchy*: *Resource* $\rightarrow$ *Resource*, which represents the containment relation between two resources. The *containsResource+* property is the transitive and reflexive closure of the inverse of *resourceHierarchy*.
- **Action**: it represents the types of operations over resources available to principals.
- **Authorization**: it represents the allowed and forbidden behaviors for a principal. Each authorization is characterized by its sign and the *Principal* it is granted to. These fundamental characteristics are represented by the two properties *sign* and *grantedTo*:
  - $sign$: $Authorization \rightarrow \{+, -\}$ assigns the sign $+$ to authorizations that grant some privileges, the sign $-$ to those that negate them. The signs are two different individuals. Every authorization must have one and only one sign, in DL formulas *Authorization* $\equiv = 1.$*sign*. For clarity, we can define two derived terms to denote positive and negative authorizations, with the following DL formulas:

$$PositiveAuthz \equiv \exists sign.\{+\}$$
$$NegativeAuthz \equiv \exists sign.\{-\}$$
$$PositiveAuthz \sqcap NegativeAuthz \sqsubseteq \perp$$

  - $grantedTo$: $Authorization \rightarrow Principal$ connects each authorization to the principal it is granted to. The *grantedTo* relation is mandatory (in DL terminology, *serial* on authorizations) and functional (with a DL formula, *Authorization* $\equiv = 1.$*grantedTo*).

Consistently with modern Role-Based Access Control models [15], authorizations are used to describe the direct privilege to act in the system and the ability to activate a role. For this reason, authorizations are refined in two subclasses:

- **System authorization:** it describes the allowed/forbidden *action* and the involved *resource*. In DL formulas, *SystemAuthorization* $\sqsubseteq$ *Authorization*, *toDo* : *SystemAuthorization* $\rightarrow$ *Action*, *SystemAuthorization* $\equiv = 1.$*toDo*, *on* : *SystemAuthorization* $\rightarrow$ *Resource*, *SystemAuthorization* $\equiv = 1.$*on*.
- **Role authorization** : it specializes *Authorization* with the specification of the *role* that the principal is allowed or forbidden to assume. In DL formulas, we write *RoleAuthorization* $\equiv$ *Authorization* $\sqcap\neg$ *SystemAuthorization*, *enabledRole* : *RoleAuthorization* $\rightarrow$ *Role*, *RoleAuthorization* $\equiv = 1.$*enabledRole*.

  Positive Role authorizations grant the ability to activate roles. For simplicity's sake, we introduce the property *canHaveRole+*: *Identity* $\rightarrow$ *Role* to represent the roles that each identity can activate, directly or indirectly, thanks to the presence of positive role authorizations.

  To compute the values of the *canHaveRole+* property, which represents direct and indirect user-role assignment,

[1]In Description Logic terminology, *role* is used to denote a property. To avoid ambiguity, in the paper we avoid this use of the term "role".

[2]The notation "$R : A \rightarrow B$" has to be interpreted according to Description Logic conventions. It states that A and B are respectively the domain and the range of the property R, with no further constraints about functionality nor completeness of R (i.e., multiple values in B can be associated with a given instance in A, and some instances in A may not be associated with a value in B). If necessary, functionality must be stated via a dedicated axiom.

we add the following axioms to the ontology *grantedTo*$^{-1}$ $\circ$ *isPositiveAuth* $\circ$ *enabledRole* $\sqsubseteq$ *canHaveRole*+, *grantedTo*$^{-1}$ $\circ$ *isPositiveAuth* $\circ$ *enabledRole* $\circ$ *canBe*+ $\sqsubseteq$ *canHaveRole*+, *containedIn* $\circ$ *canHaveRole*+ $\sqsubseteq$ *canHaveRole*+, where *isPositiveAuth* is the characteristic property of *PositiveAuthorization*[3].

In the remainder of this work, the sign of an authorization will be managed in different way for system authorizations and role authorizations. For the first, sign conflicts can be checked for conflict analysis and policy minimization. Positive role authorizations between roles correspond to role – super-role relationships, while negative role authorizations are used as a constraint, to be verified during Separation of Duty checks. Without loss of generality, we will assume that no negative role authorization is assigned to Identities.

III. Policy Incompatibility

Current space applications support a large number of users and resources, with the need to create large sets of permissions. This large size, combined with a policy model supporting both positive and negative permissions, supported by our model as by most modern access control models, makes the automatic identification of conflicts between authorizations extremely beneficial. The *Policy Incompatibility* reasoning service aims at checking whether a set of System Authorizations contains inconsistencies.

Given a policy that contains a set $A = \{a_1, \ldots, a_n\}$ of System Authorizations, we define: 1) p_i the set of $Principal$s the authorization a_i is granted to, directly or indirectly, 2) act_i the $Action$ associated with the authorization a_i, 3) r_i the set of $Resource$s to which the enabled permission can be applied, directly or indirectly. An inconsistency in the policy may arise when there exists, at least, $a_i, a_j \in A$, such that $sign(a_i) \neq sign(a_j) \wedge p_i \cap p_j \neq \emptyset \wedge act_i = act_j \wedge r_i \cap r_j \neq \emptyset$. This kind of conflict is usually called *Modality conflict* and arises when principals are authorized to do an action on a resource by a positive authorization, and forbidden to do the same action on the same resource by a negative authorization. In this case the two authorizations are said to be *incompatible*.

The management of *Modality conflicts* first requires to detect them. Then, when a conflict is detected, a choice has to be made about how the conflict can be solved, deciding between the positive and negative authorization which is the one that is going to hold. We note that it is usually impractical to simply remove the "weaker" authorization and to replace the policy with one that does not present modality conflicts. Even in a scenario like the one considered in this paper, where the containment structure of principals, actions, and resources is static, this step may lead to an excessive growth of the policy. The common approach consists in identifying a criterion to use in the resolution of conflicts, and applying it every time a modality conflict is detected.

[3]The characteristic property indicates if a node belongs to a given class. In our case it can be defined using the following axioms: *isPositiveAuth*: *PositiveAuthz* $\rightarrow$ *PositiveAuthz*, $\top \sqsubseteq \leq 1.$*isPositiveAuth*, *PositiveAuthz* $\equiv \exists_{self}$*isPositiveAuth*

In the literature and in systems, several criteria have been proposed and implemented to solve this kind of conflict [13]. A simple criterion is the "*Denials take precedence*", which states that, in case of conflicts, the Negative Authorization always wins. The "*Denials take precedence*" criterion is a special case of techniques based on explicit priority assignments, i.e., in which negative authorizations have the highest priority. Our approach can be easily extended to handle explicit priorities by using SWRL built-ins. A more flexible criterion is the "*Most specific Wins*", which states that, when one authorization is less specific than the other, the more specific one wins (i.e. the authorization that applies to the smaller set of individuals in our model wins).

For simplicity we introduce the property *canActAs* to take into account the different ways a principal can activate the profile of another principal. *canActAs*: *Principal* $\rightarrow$ *Principal* is a transitive and reflexive property such that *containedIn* $\sqsubseteq$ *canActAs*, *canBe*+ $\sqsubseteq$ *canActAs* and *canHaveRole* $\sqsubseteq$ *canActAs*. *canActAs*(p_1, p_2) means that p_1 is more specific than p_2.

We have also introduced the property *winVs*: *SystemAuthorization* $\rightarrow$ *SystemAuthorization* which can be used to express the fact that the conflict involving the authorizations a_1 and a_2 is solved by applying a_1.

In order to identify the modality conflicts in a policy, we use SWRL rules. SWRL rules are a particular kind of implication, which can be expressed using a subset of the OWL language. A SWRL rule is composed by an antecedent and a consequent, and each of them is composed by zero or more atoms, i.e., OWL properties, classes or predefined built-in expressions. Each atom can refer to individuals using variables, denoted by a question mark as a prefix. The semantics is that when the antecedent of the rule is true, then also its consequent must be true.

We can detect modality conflicts and compose them following the "*Most specific wins*" criterion, using SWRL rules like the following:

```
on(?a1,?r1), toDo(?a1,?act),
    grantedTo(?a1,?pr1), sign(?a1,?s1),
    on(?a2,?r2), toDo(?a2,?act),
    grantedTo(?a2,?pr2), sign(?a2,?s2),
    containsResource+(?r2,?r1),
    canActAs(?pr2,?pr1),
    DifferentFrom(?s1,?s2),
    DifferentFrom(?pr1,?pr2) ->
    winsVs(?a2,?a1)
```

When the conflict involves two policies that have the same specificity level, we apply the "*Denials take precedence*" criterion, with the following rule:

```
PositiveAuthz(?a1), SystemAuthorization(?a1),
    on(?a1,?r), toDo(?a1,?act),
    grantedTo(?a1,?pr), NegativeAuthz(?a2),
    SystemAuthorization(?a2), on(?a2,?r),
    toDo(?a2,?act), grantedTo(?a2,?pr) ->
    winsVs(?a2,?a1)
```

However, not all the conflicts can be solved using these criteria, because in some conflicts we can not find a policy that is more specific than the other one (e.g., one has a higher level in the resources and the other in the principals). To detect these cases, which we assume have to be notified to the security administrator responsible for the design of the policy, we defined a new symmetric property *unsolvableConflict*: *SystemAuthorization* $\rightarrow$ *SystemAuthorization*, which contains all the pairs of policies involved in conflicts that are not solvable using the criteria presented above.

The *Policy Incompatibility* service produces three sets of authorizations: 1) a set of authorizations that does not contain any conflict, 2) a set of conflicting authorizations, in which all the conflicts are solved using the "*Most specific wins*" and "*Denials take precedence*" criteria, 3) a set of unsolved conflicts.

IV. Separation of Duty

A common class of constraints represented in security policies is *Separation of Duty* (SoD). These constraints follow the common best practice for which sensitive combinations of permissions should not be held by the same individual in order to avoid the violation of business rules. *Role Based Access Control* is particularly well-suited to express this kind of constraints, because the role hierarchy permits an easy mapping of real world business rules to the access control model. In the following, we primarily consider *Role-based Static SoD*. Given two roles $role_1$ and $role_2$, the Static SoD between these two roles means that there must not exist a user u who can activate both $role_1$ and $role_2$. We focus on the Static SoD, but the model can be immediately extended to support also Dynamic SoD constraints, where it is required to check that a user does not concurrently activate conflicting roles (it is sufficient to assume that the model provides the concept of *active* role). The aim of the *SoD Satisfiability* service is to check whether a set of authorizations satisfies a set of SoD constraints.

In our model the roles are represented as individuals of the class $Role$, the role hierarchy is expressed using the property *roleHierarchy*, and the user-role assignments are represented using the Role Authorizations introduced in Section II, which allow a user to enable a specific role.

We can express a SoD constraint between the role $role_1$ and the role $role_2$ by using a negative role authorization $auth_1$, such that *enabledRole*$(auth_1, role_1)$, *grantedTo*$(auth_1, role_2)$. The authorization $auth_1$ denies $role_2$ to activate $role_1$ and $role_1$ to activate $role_2$. Before analyzing a policy to detect SoD conflicts we execute a normalization step, which replaces all the role authorizations between roles with semantically equivalent *roleHierarchy* properties in order to simplify the analysis. A role authorization that assigns a role $role_2$ to a role $role_1$ is equivalent, for SoD satisfiability analysis, to a super-role relation between $role_1$ and $role_2$ and, thus, to *roleHierarchy*$(role_1, role_2)$.

Separation of duty constraints have to be enforced both at role hierarchy level, in this way we directly prevent that a role $role_1$ is declared super-role of another role $role_2$ such that $role_1$ and $role_2$ are in a SoD constraint, and at user hierarchy level, to avoid that to a user are assigned, directly or indirectly, two roles $role_1$ and $role_2$ that are involved in a SoD constraint. In order to keep track of all SoD conflicts on *Roles*, we have defined a class *SoDConflictOnRole* $\sqsubseteq$ *Role*. Separation of duty constraints on the role hierarchy can be expressed adding to the ontology an axiom in the form:

$$SoDConflictOnRole \equiv \exists canBe+.\{role_1\} \sqcap \exists canBe+.\{role_2\}$$

for each *RoleAuthorization* $auth$ such that $sign(auth, -)$, $grantedTo(auth, role_1)$, $enabledRole(auth, role_2)$. We can thus enforce the SoD at role hierarchy level simply adding the axiom *SoDConflictOnRole* $\sqsubseteq \bot$ to the ontology.

In a similar way to what we have done for the identification of SoD conflicts at role hierarchy level, we defined a class *SoDConflictOnId* $\sqsubseteq$ *Identity* that keeps track of the conflicts on the user hierarchy. We express SoD constraints by adding to the ontology an axiom in the form:

$$SoDConflictOnId \equiv \exists canHaveRole+.\{role_1\} \sqcap \exists canHaveRole+.\{role_2\}$$

for each *RoleAuthorization* $auth$ such that $sign(auth, -)$, $grantedTo(auth, role_1)$, $enabledRole(auth, role_2)$. In order to enforce the SoD constraints we simply have to add to the ontology the axiom *SoDConflictOnId* $\sqsubseteq \bot$.

Our approach can be easily extended to handle other kinds of SoD constraints, such as *Permission-based SoD* (which requires that no user is allowed to do both actions a_1 and a_2) or *Object-based SoD* (which requires that no user can access both resources res_1 and res_2), as shown in [7].

V. Implementation and Experiments

We implemented a prototype for the evaluation of the performance of the techniques presented in this paper. The prototype consists of two Java modules that invoke the *Policy Incompatibility*, and *Separation of Duty Satisfiability* services. The prototype uses the API provided by the OWL API library [8] to access both the ontology and the SWRL rules and it uses Hermit [16] as reasoner. There are no freely available large datasets of real security policies, neither for space applications nor for generic information systems. For this reason, we chose to test our prototype against policies built according to an interpretation of the data in bibliographic databases. We used randomly selected subsets of *PubMed Central*[4] (PMC), well known in the medical sciences, and DBLP[5], well known in the computer science community. Each of them provides a rich set of attributes and relationships that represent real and extensive social networks (as a parallel to space applications, we can consider a single paper/resource to correspond to the description of sensor data about a specific region or a specific sensor, with a number of users interested in accessing it and responsible for manipulating that information). PMC has rich information about journals, with a description of editorships

[4]http://www.ncbi.nlm.nih.gov/pmc/
[5]http://dblp.uni-trier.de/

and the funding of papers. DBLP has a rich description of conferences. The two databases support different experiments.

For the experiment on the *Policy Incompatibility* service, we use randomly generated subsets of the bibliographic database *PubMed Central*. We consider 3 actions: *write*, *review*, and *read*.

We create a *SingleIdentity* for each author/editor and a *Group* for each group of authors/editors in a paper/journal. For each journal issue/conference, we create an "*editor*" *Role* and an "*author*" *Role*. We also create the role authorizations that assign to each author/editor the proper role. We then created the following authorizations with identities as principals: (a) for each paper author we create the authorizations to *read* and *write* the paper and the negative authorization to *review* the paper, (b) for each editor of the issue of the journal containing the paper we add the authorizations to *read* and *review* the paper and the negative authorization to *write* the paper, (c) for each author that receives funding from the same grant that funded the paper, we add a negative authorization to *review* the paper and an authorization to *read* the paper, (d) for each group representing the institution to which the author is affiliated, we add the authorization to *read* the paper, (e) for each group representing the editorial board of the journal that published the paper we add the authorizations to *read* and *review* the paper and the negative authorization to *write* the paper.

With this model we are able to create a rich set of authorizations that is associated with the structure of a concrete application. The conflicts detected would correspond to anomalies that relate with possible conflicts of interest.

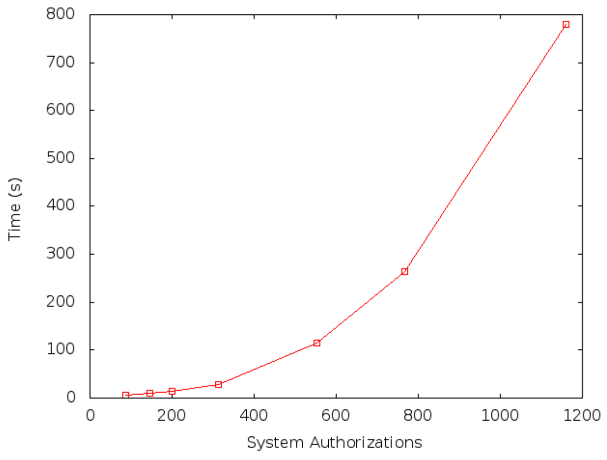
Finally, the *Separation of Duty* (*SoD*) service solves the problem presented in Section IV. We enforced the *Separation of Duty* constraint on author and editor roles for the same conference/journal issue.

A. Experimental results

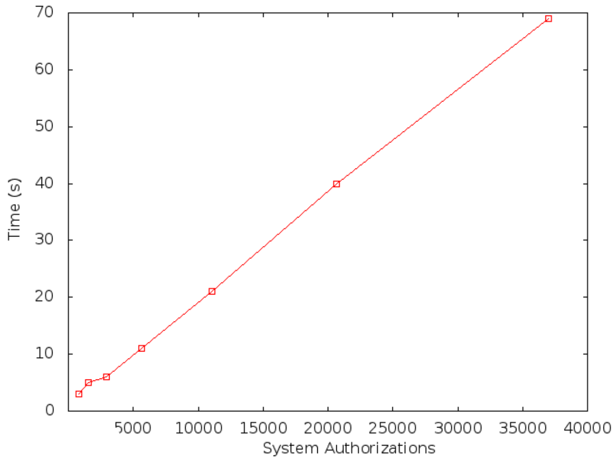
Experiments have been run on a PC with two Intel Xeon 2.0GHz/L3-4MB processors, 8GB RAM, four 1-Tbyte disks and Linux operating system. The results of the experiments are reported in the charts in Figure 1. Each observation is the average of the execution of ten runs.

The first set of experiments aimed at evaluating how the performance of the *Policy Incompatibility* service evolves with the increase in the number of authorizations. Figure 1(a) reports the observed performance. It is clear that this solution is applicable for policies with a relatively small number of authorizations. For large policies, the simplicity in the definition of the verification rules is associated with a large computational cost.

For the largest policies considered in our experiments, a significant benefit can be obtained from the use of tools, like the reasoners able to efficiently process DL structures, which require a different and more complicated representation of the problem. In [7] we present how the *Policy Incompatibility* service can be mapped to the *Separation of Duty* service, thus using efficient DL structures.



(a) Policy Incompatibility performance.



(b) Separation of Duty analysis performance.

Figure 1. Experimental results

The second experiment aimed at evaluating the performance of the *Separation of Duty* service. We found that, on average, 0.77% of the role authorizations in a policy break the SoD constraints.

In Figure 1(b) we can see how the performance evolves with the increase of the number of role authorizations. Even for extremely large policies (more than 35,000 role authorizations), the response time remains adequate for the profile of the security design activity, which operates offline in sessions that have long duration.

VI. Related Work

The management of security policies for space applications has not been considered explicitly in past literature and we have to refer to classical work on the analysis of security policies, which saw a variety of approaches (e.g., outside of the Semantic Web scenario we note [3], [4]). A few works have considered the application of Semantic Web technologies for policy analysis, in order to obtain advantages from the use of reasoners [6], [11]. An important observation is that the works on the use of Semantic Web Technology for policies rarely show empirical results going beyond simple toy examples. Only in [11] appears an analysis of a small set of ontologies, still without reaching the policy sizes we consider in our experiments. The approach that builds a policy starting form the consideration of bibliographic databases has been successfully applied in previous work [5], leading to the realization of representative large security policies.

Finin et al. [6] use OWL to formalize Role Based Access Control (RBAC). They propose two approaches: 1) the representation of roles by means of OWL classes, 2) the representation of roles as values, in a similar way to our approach. In the first approach they enforce SoD constraints by means of the *owl:disjointWith* property, while in the second approach they express constraints by means of N3 rules. Their work offers only the *SoD Satisfiability* service, while our work considers a richer collection of services. They do not show experimental results over large policies; since N3 rules are similar to SWRL rules, based on our experience we expect that their approach, which considers roles as individuals, would probably show poor scalabity. Their approach also supports attribute based access control (ABAC); the similarity with our base model makes us confident that our approach can be easily extended to handle ABAC.

In [11] Kolovski et al. tackle the fact that complex XACML policies are hard to understand and evaluate manually, and, thus, automated tools are needed. They propose to map XACML to DL, to benefit from off-the-shelf DL reasoners. They identify several reasoning services on policies: (1) *Constraint Analysis*, (2) *Policy Comparison*, (3) *Policy Verification*, (4) *Policy Incompatibility*, (5) *Policy Redundancy*, (6) *Policy Querying*. Compared to this approach, our work provides an implementation of the most significant of those services using Semantic Web technology, and this allows us to use a larger set of tools, such as SWRL or SPARQL, which offer a more direct and easy realization of the services. Another difference is that our model can express user and resource hierarchies and, thus, it can represent more complicated scenarios.

Some works extend standard access control models by adding new kinds of policies, i.e. resiliency policies [12] and obligations [2]. Although these kinds of policy are interesting from a research point of view, they are not currently implemented in industrial solutions. This is the main reason why our tool does not yet handle them. We plan to extend our approach in order to support these policies.

Detection of anomalies in ACLs is an interesting topic also in firewall configurations and management. Although several works [1], [17] recognize the importance of anomaly detection in order to increase the performance of firewalls and reduce the errors in the security configuration, the models used in these works are simple, they usually have only two actions, i.e. *accept* and *deny*, and they consider only the hierarchy of IP addresses, and thus cannot be directly compared with approaches that consider more complex access control models.

VII. Conclusions

Information systems are becoming increasingly difficult to manage, with the integration of resources controlled by different actors, and access offered to a large variety of users. Service oriented architectures facilitate this evolution. We presented approaches that enhance the management of security policies in these scenarios. The techniques we described are built on several of the tools provided by the *Semantic Web*. A trade-off has been identified between simple solutions that use a direct model and rely on SWRL rules, with limited scalability, and approaches that require a little effort to adapt the models to the application of reasoners like HermiT, offering good performance and scalability. Optimizations prove to contribute a significant benefit. In general, the work presented offers a demonstration that the *Semantic Web* can have an important role in the construction of techniques for security policy analysis in large and integrated information system. In general, the work presented offers a demonstration that the *Semantic Web* can have an important role in the construction of techniques for security policy analysis in large and integrated systems.

References

[1] E. Al-Shaer and H. Hamed. Modeling and management of firewall policies. *IEEE Trans. on Network and Service Management (TNSM)*, 1(1), 2004.

[2] C. Bettini, S. Jajodia, X. S. Wang, and D. Wijesekera. Provisions and obligations in policy management and security applications. In *Proc. of VLDB 2002*.

[3] V. Ciriani, S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati. Combining fragmentation and encryption to protect privacy in data storage. *ACM Trans. on Information and System Security (TISSEC)*, 13(3), 2010.

[4] S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati. Assessing query privileges via safe and efficient permission composition. In *Proc. of CCS 2008*.

[5] S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati. Encryption policies for regulating access to outsourced data. *ACM Trans. on Database Systems (TODS)*, 35(2), 2010.

[6] T. Finin, A. Joshi, L. Kagal, J. Niu, R. Sandhu, W. Winsborough, and B. Thuraisingham. R OWL BAC: representing role based access control in OWL. In *Proc. of SACMAT 2008*. ACM.

[7] M. Guarnieri and E. Magri. Techniques for conflict detection and minimization for access control policies, Master Thesis, Università degli Studi di Bergamo, 2012.

[8] M. Horridge and S. Bechhofer. The OWL API: A Java API for OWL ontologies. *Semantic Web*, 2(1):11–21, 2011.

[9] I. Horrocks and P. Patel-Schneider. Reducing OWL entailment to description logic satisfiability. *Journal of Web Semantics*, 2004.

[10] W. C. . http://www.w3.org/TR/owl ref/. *OWL Web Ontology Reference*.

[11] V. Kolovski, J. Hendler, and B. Parsia. Analyzing web access control policies. In *Proc. of WWW 2007*. ACM.

[12] N. Li, M. V. Tripunitara, and Q. Wang. Resiliency policies in access control. In *Proc. of CCS 2006*. ACM.

[13] E. Lupu and M. Sloman. Conflicts in policy-based distributed systems management. *IEEE Trans. on Software Engineering (TSE)*, 25(6), 1999.

[14] P. Samarati and S. De Capitani di Vimercati. Access control: Policies, models, and mechanisms. In *Foundations of Security Analysis and Design*, volume 2171, pages 137–196. Springer, 2001.

[15] R. Sandhu. Role-based access control. *Advances in computers*, 46, 1998.

[16] R. Shearer, B. Motik, and I. Horrocks. HermiT: A highly-efficient OWL reasoner. In *OWLED*, 2008.

[17] L. Yuan, H. Chen, J. Mai, C. Chuah, Z. Su, and P. Mohapatra. FIREMAN: a toolkit for firewall modeling and analysis. In *Proc. of IEEE S&P*, 2006.