A Role and Attribute Based Access Control System Using Semantic Web Technologies*

Lorenzo Cirio[1], Isabel F. Cruz[1], and Roberto Tamassia[2]

[1] University of Illinois at Chicago
{lcirio|ifc}@cs.uic.edu
[2] Brown University
rt@cs.brown.edu

Abstract. We show how Semantic Web technologies can be used to build an access control system. We follow the role-based access control approach (RBAC) and extend it with contextual attributes. Our approach provides for the dynamic association of roles with users. A Description Logic (DL) reasoner is used to classify both users and resources, and verify the consistency of the access control policies. We mitigate the limited expressive power of the DL formalism by refining the output of the DL reasoner with SPARQL queries. Finally, we provide a proof-of-concept implementation of the system written in Java.™

1 Introduction

In this paper, we present an access control system for context-aware environments designed and built using Semantic Web technologies. We adopt the *role based access control* (RBAC) model [12]. In an RBAC system, roles are assigned to users statically using a procedure performed by the security administrators. Although revisions are possible, they are not supposed to be frequent nor to be done at run time [12]. This approach can be restrictive in various situations, including in mobile situations that are common in *context-aware* or *pervasive computing*, where the identity of the users is not known in advance. For example, we may wish to grant a role to a visitor, or to a first time client, without permanently registering the client's data.

Recently, various proposals have appeared in the literature to extend RBAC with concepts like team membership, users' tasks, organizational hierarchy, and contextual information (such as position and time) [16]. Each of these approaches captures an interesting (yet partial) aspect to be directly integrated in the access control system.

To introduce flexibility into the procedure of role assignment, we borrow ideas from attribute-based access control systems (ABAC) [1, 19]. In an ABAC system, permissions are associated with a set of rules expressed on measurable parameters and are granted to users who can prove compliance with these rules. However, unlike ABAC, we do not directly associate the permissions with the

* Work supported in part by NSF grants IIS-0326284, IIS-0324846, IIS-0513553, IIS-0713403, and OCI-0724806.

attributes, but instead borrow the concept of role from RBAC, which we use as an intermediate structure between attributes and permissions.

In an RBAC system, the first interaction between the user and the system is an identification procedure, followed by the retrieval of the roles leased to the user from a database. We replace this phase with a handshake procedure in which the roles that can be claimed by the user are determined on the basis of the provided attributes. These roles are then enabled as usual in the activation phase.

Using this approach, we can move from an authentication system based just on identity to one that takes into account attribute values. For example, in the case of our visitor, a credential issued by a trusted third party or GPS reading can supply the value for an attribute that will give access to a particular role.

We summarize the contributions of our work as follows:

Access model. By combining RBAC, which supports static roles, with ABAC, we introduce a new access control model that enables dynamic assignments of roles to users in two different ways. First, privileges associated with resources are dynamically assigned depending on the attribute values of the resources. Second, attribute values associated with users determine the association of users with privileges. This kind of approach is especially suited to context-aware or pervasive computing.

Semantic Web technologies. We develop a framework that supports our security model using Semantic Web technologies and in particular OWL-DL. To this end, we produce a high level OWL-DL ontology that expresses the elements of a Role Based Access Control system, and build a domain specific ontology that captures the features of the application. Inferencing as supported by OWL-DL determines containment among classes, for example, how policies relate to resources or how instances are classified into their correct categories.

Expressiveness and design. We encountered several expressiveness problems and design choices. In particular, one limitation of the actual OWL classifier is the limited or missing support of concrete data types, which prevents stating conditions that involve data type comparison. To address this problem, we use SPARQL queries to express path properties. We use ontology design "best practices" [11] for developing complex structures in OWL.

Implementation. We provide a proof-of-concept implementation of the system, integrating standard elements with our application specific code. Our code is written in Java.™ The RDF data are processed using the Jena™ framework, which also provides a SPARQL query engine. In addition, our system does not depend on a specific DL-reasoner. Therefore, by leveraging existing technologies such as secure network infrastructures, RDF data processors, and DL-reasoners, our implementation can quickly adapt to new standards and new implementations as they become available.

This paper is organized as follows. In Section 2 we describe an OWL-DL ontology that expresses the elements of a Role Based Access Control system to define privileges and associate them with roles and resources in a domain ontology. In Section 3 we describe briefly the OWL design patterns we use to

build the ontologies and the software architecture of our implementation. We discuss related work in Section 4 and show how our work advances the state of the art when compared to that work. Finally, we draw conclusions and point to future work in Section 5.

2 RBAC Modeling

In this section, we describe our method for creating an OWL-DL ontology that expresses the modeling abstractions of RBAC. We then show how we attach this ontology to a domain ontology and explain the tasks that are performed by the security administrator and by the DL classifier. We also discuss the limitations in expressiveness of OWL-DL, the use of SPARQL queries, and how we integrate SPARQL with OWL-DL.

In the development of the RBAC ontology, we have followed these principles: (1) the access control ontology should limit itself to expressing the modeling abstractions of RBAC; (2) no hypothesis is made about the domain knowledge, neither semantically nor syntactically—the resulting role ontology shall not depend on external factors like the type of organization or type of procedures nor on the structure of the domain ontology (which could be either monolithic or a layered system composed of different ontologies); and (3) we do not capture any workflow procedure inside the model so as to preserve both simplicity and generality.

In agreement with these hypotheses, we provide a tool that can express RBAC constructs, which are intended to be imported and used by the domain ontology, such as the library ontology of Figure 1. In this figure, we represent three ontologies: the RBAC ontology and two domain ontologies: *general* and *custom*. The former is an "off-the-shelf" (reused) ontology and the latter an ontology specifically developed for the access control system. Higher level abstractions or extensions can be implemented outside of the role ontology, either directly in the domain ontology or in additional, intermediate layers.

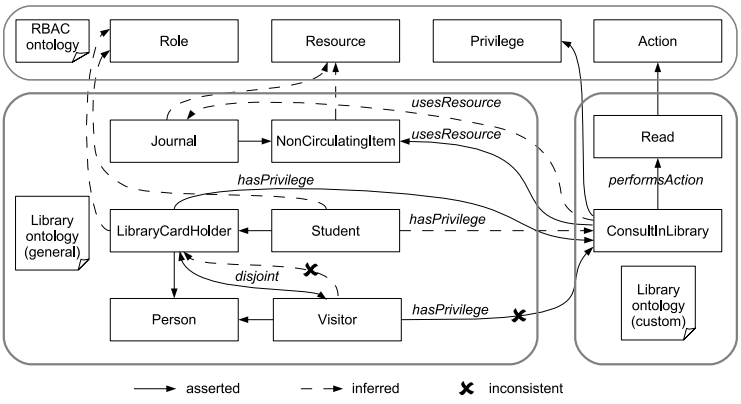

Fig. 1. Relationships between the domain ontologies and the RBAC ontology.

In providing an RBAC classification for all the classes of the domain ontologies, there are two actors at play: the security administrator and the DL classifier. We assume that the security administrator explicitly defines: (1) privileges (e.g., $ConsultInLibrary = (NonCirculatingItem, Read)$); (2) the association of privileges with classes in the domain ontology (e.g., *ConsultInLibrary* with *LibraryCardHolder*). The DL classifier propagates the RBAC elements according to the axioms that were originally stated in the domain ontologies using any valid inference, for instance, inheritance (e.g., every subclass of role *Student* will be classified as *Role* and will enjoy the privileges of *Student*).

The RBAC ontology has the following four classes:

Action. This is a partial, or self-standing, class that represents an action that can be performed by a user on a resource. It is intended to be the super class of the actual actions that can be performed in the system (e.g., *Read*).

Resource. This is a defined class, representing the authorization objects. The DL classifier will place under *Resource* all the classes that match the condition

$$\exists subject\_to.Privilege \sqcap \neg\{Action, Role, Privilege\}$$

that is, every instance that has relationship *subject_to* with an entity of type *Privilege* and that is not an *Action*, a *Role*, or a *Privilege*. Using the classifier, we can identify all the objects that have been treated like a *Resource* in the domain ontology. In our example of Figure 1, this is the case for *Journal* and *NonCirculatingItem*.

Privilege. This is a partial (self standing) class representing a pair (a, r) with $a \in Action$, $r \in Resource$. The one-to-one association is imposed by a cardinality constraint. In our example, privilege *ConsultInLibrary*, represents the pair *(NonCirculatingItem, Read)*. By using inheritance between *Journal* and *NonCirculatingItem*, the DL classifier will also infer the privilege *(Journal, Read)*.

Role. This is a defined class. The DL classifier will place under *Role* all the classes that match the condition $\exists hasPrivilege.Privilege$. We then expect that any class that is declared to have a privilege in the domain ontology to be classified as *Role*, as is the case with *Student* and *LibraryCardHolder*.

The RBAC ontology has the following properties:

$hasPrivilege \subset Role \times Privilege$. It is a many-to-many association of roles and privileges. Figure 1 shows how the property is imported into a domain ontology. It is used to indicate that a specific class in the domain (e.g., *Student*) has some privileges (e.g., *ConsultInLibrary*) and therefore should be considered as a *Role*, as already mentioned. We are using the characteristic of Description Logic that states that range and domain are not considered as constraints to be checked, but as axioms conveying additional information. Therefore, by establishing that a class (e.g., *Student*) is associated with some subclass of *Privilege* (e.g., *ConsultInLibrary*) through *hasPrivilege*, we are indicating that this class is actually a subclass of *Role*.

This approach lets the security administrator define roles simply by attaching property *hasPrivilege* to the appropriate classes in the domain of interest. The

static separation of duties is formulated in the domain ontology by declaring those classes disjoint. For instance, our library ontology ensures that *Visitor* and *LibraryCardHolder* are disjoint roles. Therefore, a user cannot be associated with these two roles simultaneously and can only enjoy the privileges that go with one of them.

$notTogetherWith \subset Role \times Role$. It is a many-to-many association of roles with roles, used to express the dynamic separation of duties. Therefore if *RoleA* is active, the system will refuse to activate any instance of a subclass of *Role* that is in a *notTogetherWith* relationship with *RoleA*. The property is declared to be symmetric, therefore the DL classifier will compute the symmetric closure of the relation, ensuring that two roles are dynamically separated even when just one of them is declared incompatible with the other.

$performsAction : Privilege \rightarrow Action$. It is a total function that associates each privilege with the action it allows to perform. Together with *usesResource*, this construct is used to ensure that each privilege is a pair (a, r) with $a \in Action$ and $r \in Resource$.

$usesResource : Privilege \rightarrow Resource$. It is a total function associating to each privilege the resource on which it operates. Together with *performsAction*, this construct is used to ensure that each privilege is a pair (a, r) with $a \in Action$, $r \in Resource$. The inverse of *usesResource* is a property named *subject_to*. It is not a function because one resource can be managed by different privileges.

An issue that we must address is that of limitations of expressiveness that arise in OWL-DL. Consider the example where we want to state that a *Candidate* is a *Student* that prepares a *Thesis* and is advised by an *Adviser*:

$$Candidate \equiv Student \sqcap \exists prepare.Thesis \sqcap \exists is\_advised.Adviser$$

Likewise, we can state that an *Adviser* advises a *Candidate* and reviews a *Thesis*:

$$Faculty\_Member \sqcap \exists review.Thesis \sqcap \exists advise.Student$$

These statements only involve classes, therefore there is no way for us to specify that the actual instance of thesis prepared by a specific student is the same instance of thesis that is reviewed by the intended adviser [7]. Figure 2 shows how the model described is interpreted by the DL reasoner. The classifier will infer that the instance of *Adviser* has to be linked to an instance of *Thesis*, but cannot infer that this instance is the one linked to the student who is supervised by the adviser.

A possible solution to this problem is to first apply a DL classification and then a set of Horn clauses [17]. However, our problem of verifying path properties between instances can be reduced to querying the ontology to verify if such a path exists. Queries expressed in SPARQL provide us with the solution to this problem. The language adopts a closed world assumption, following the usual convention for database systems. This is in contrast with the open world assumption adopted by Description Logic that does not imply that there exist particular instances of the classes involved that satisfy a particular constraint.

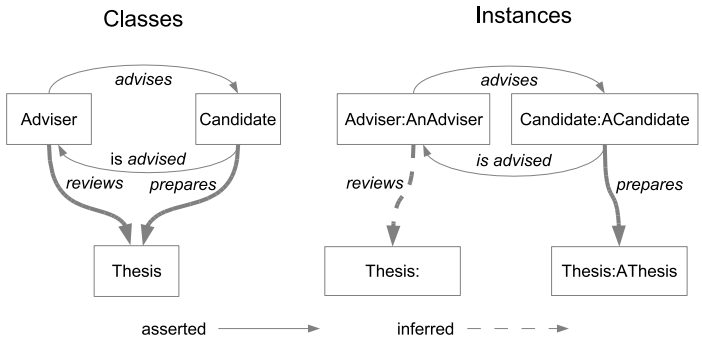

Fig. 2. An ontology showing a path property.

In particular, we are interested in ensuring that some constraints expressed at design time are met at run time, therefore the `ASK` construct fits our needs. The query of Figure 3 is the path query that solves the example previously discussed and illustrated in Figure 2. The `WHERE` section of the query is used to specify the graph pattern that we are looking to match. Alternative paths can be taken into account, using the keyword `UNION` to indicate them.

```
ASK WHERE {
   ACandidate
      prepares ?thesis;
      is_advised AnAdviser.
   AnAdviser reviews ?thesis
}
```

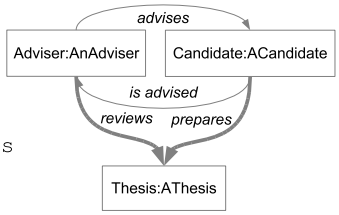

Fig. 3. Path property expressed in SPARQL.

In expressing queries, there is information that could be interesting to refer to, such as the session identifier, which is known only at run time. To overcome this limitation, we introduce the following convention: the variable `?session` can be used as a reference to the session identifier. At run time, the execution environment will take care of replacing the symbol with its actual value, wherever it appears.

We seamlessly integrate SPARQL queries with our OWL ontology using annotation properties, which carry meta-information about the ontology. For example, the standard defines the property *owl:versionInfo* to trace the evolution in time of an ontology or *rdfs:label* to provide a human-readable tag for a node.

We have declared two annotation properties, *requiresTrue* and *requiresFalse*, with domain *Role* and range *String*. They can be used in the domain ontology to specify additional constraints that have to be checked by the runtime envi-

ronment. SPARQL queries such as those we have just introduced, which return a Boolean value, will be the string pointed to by a property.

3 Implementation

In this section, we describe briefly our prototype implementation, focusing on OWL design patterns and software architecture. We use three OWL design patterns: value partition, linked list, and recursive composition, which are summarized below.

Value partition. This OWL design pattern addresses the problem of assigning values to attributes of a class and is recommended by the W3C Semantic Web Best Practices and Deployment Working Group [11].

Linked list. Inspired by Drummon *et al.* [6] and leveraging the reasoning capabilities of OWL, we have developed this OWL design pattern to represent a sequence of nodes connected by links.

Recursive composition. Adapting the composite design pattern in software engineering, we have developed this OWL design pattern to model a containment tree without using the subclassing mechanism. The navigation from child to parent is given by property *containedIn*, which is declared to be transitive. To avoid unexpected inferences, we add an axiom that limits the containment to instances of a parent class [14].

Our access control system is based on the following assumptions:

- The background knowledge used to make the access decisions is available as Semantic Web ontologies. Several techniques exist for converting relational databases and XML databases into ontologies (see, e.g., [4, 20]).
- The organization has established a secure infrastructure to register, propagate, request and verify user attributes. This functionality is offered by network services that support single sign-on across domains and trusted user directories. See, for example, PERMIS [2] and Shibboleth [13].
- We use the ontology format also for the attributes. Again, conversion techniques can be applied if the native format is different.

The interaction with the system is performed through a Java API and is based on web standards. Logically, it consists of the following phases: (1) configuration of the system by loading the ontologies; (2) initialization and consistency verification of the ontologies by the DL reasoner; (3) instantiation of sessions associated with access requests. The session manager acts as a policy enforcement point(PEP). The policy decision point (PDP) is implemented with code that wraps the DL-reasoner.

The code is organized into classes associated with the elements of the RBAC ontology. Additional functionality is introduced for the purpose of managing the sessions. The library is completed by several helper classes dealing, for example, with cryptographic operations and the management of name spaces. We leverage the Jena^{TM} library freely available from HP.

4 Related Work

Neumann and Strembeck have designed and implemented *xoRBAC*, a network service offering role based access control [10]. They also extend this service to take into account contextual information, which is fed into the system via software sensors whose output can enter into the formulation of the policies [15]. Users are authenticated using digital certificates in standard X.509 format. The RBAC information is stored in RDF-XML format, but the authors never mention the use of reasoners, therefore we assume that RDF is only used as a way to store data. The policies are written in a dedicated language that is later converted into an XOTcl (eXtended Object Tcl) script and executed by the system.

Kagal *et al.* propose *Rein*, a distributed framework based on ontologies to share and compose access control policies [8]. They reuse reference policies [9] and adapt them to their needs. While policies may be expressed in different ontology languages, their prototype uses the rule language N3 and the reasoner CWM. The authors motivate the choice of a rule based language over a DL-based language, with a higher expressive power. They note that this choice presents a couple of difficulties as it prevents classifying the different policies and detecting incompatibilities in their definitions.

Toninelli *et al.* describe a context-aware access control framework for pervasive computing [17]. They present a scenario where participants from different organizations attend a meeting. In their approach, users can safely exchange information and share resources, without knowing their identities beforehand. This method relies on sensors that capture the contextual information. Context is represented with a context ontology, which needs to be imported and is specialized from the ontologies modeling the domain of interest. Description Logic is used to classify the context models and discover their relationships (such as equivalence or generalization). The context models developed at design time need to be associated with the data from the sensors to form enforceable policies. This instantiation procedure depends on rules expressed in logic programs. As compared to our approach, they do not provide the decoupling capability that is achieved by having roles.

Di *et al.* [5] represent RBAC entirely in Description Logic. Thus, their access control model is entirely contained in a DL reasoner. They offer constructs to model the role hierarchy, the static and dynamic separation of duties and some form of cardinality constraints. They do not enrich the system with additional languages and are therefore limited to the constraints of open world reasoning. Our approach contrasts with this one because by adding another language, we are not limited to the expressiveness of a DL-reasoner.

KAoS is a rich component-based framework, for expressing, administrating and enforcing policies [18]. It is especially targeted to distributed computing environments as grid computing and Semantic Web Services. It offers a high level ontology that explicitly provides constructs to model processes and transactions. To offer this expressive power, the system integrates OWL-DL with the SWRL rule language. While our approach leverages existing technologies and

can quickly adapt to new standards and new implementations as they become available, KAoS relies on the construction of many of the system components.

Damiani *et al.* propose *GEO-RBAC*, a formal framework for "spatially-aware" RBAC for location-based applications, where roles are activated based on the position of the user [3]. The framework uses spatial entities to model objects, user positions, and geographically bounded roles and further consider hierarchies to model permission, user, and activation inheritance. Properties concerning satisfiability, implications, and evaluation of their proposed classes of constraints are proved. While role-based, GEO-RBAC does not use Semantic Web technologies.

5 Conclusions and Future Work

We have shown how available Semantic Web technologies, namely OWL and Description Logic, can be used to build an access control system. To this end, we have developed a high level OWL-DL ontology that expresses the elements of a role based access control system and we have built a domain-specific ontology that captures the features of a sample scenario. Finally, we have joined these two artifacts to take into account attributes in the definition of the policies and in the access control decision. The use of attributes is twofold: to classify users into access control roles and to classify resources as access control objects.

We have used Description Logic to express inferences, for example, to detect the containment among classes that show how policies relate to resources. The limited expressive power of Description Logic has been mitigated by introducing SPARQL queries that can check additional constraints against an available knowledge base. Thanks to the annotation properties offered by OWL, we have embedded the SPARQL queries into the ontologies, integrating the policies in one entity. Our system prototype uses Java™and can be easily extended and integrated in a networked architecture to be offered as a service.

The forthcoming OWL 1.1 standard will allow us to simplify and add expressive power to our system. In the area of security, the effectiveness of rules to express policies is demonstrated by the many existing access control systems based on this mechanism. However, while rule-based languages are widely debated in the Semantic Web community, their integration with Description Logic is still an open issue. Therefore, it is interesting to investigate how supporting a rule language will change our framework, especially considering that rule languages have complementary strengths and weaknesses with respect to Description Logic. Finally, the task of the ontology developer will be made easier by improvements to the ontology debugger. In particular, it would be interesting to increase the precision in identifying the axioms that cause an inconsistency.

References

1. M. A. Al-Kahtani and R. S. Sandhu. Induced role hierarchies with attribute-based RBAC. In *8th ACM Symposium on Access Control Models and Technologies (SACMAT)*, pages 142–148. ACM, 2003.

2. D. W. Chadwick and A. Otenko. The PERMIS X.509 role based privilege management infrastructure. *Future Generation Computer Systems*, 19(2):277–289, 2003.
3. M. L. Damiani, E. Bertino, B. Catania, and P. Perlasca. GEO-RBAC: A spatially aware RBAC. *ACM Trans. on Information and System Security*, 10(1):2, 2007.
4. C. P. de Laborda and S. Conrad. Bringing relational data into the Semantic Web using SPARQL and Relational.OWL. In *3rd Int. Workshop on Semantic Web and Databases (SWDB)*. IEEE, 2006.
5. W. Di, L. Jian, D. Yabo, and Z. Miaoliang. Using semantic web technologies to specify constraints of RBAC. In *6th Int. Conf. on Parallel and Distributed Computing Applications and Technologies (PDCAT)*, pages 543–545. IEEE, 2005.
6. N. Drummond, A. Rector, R. Stevens, G. Moulton, M. Horridge, H. H. Wang, and J. Seidenberg. Putting OWL in order: Patterns for sequences in OWL. In *OWL: Experiences and Directions (OWLED) ISWC Workshop*, 2006.
7. I. Horrocks, O. Kutz, and U. Sattler. The even more irresistible SROIQ. In *10th International Conference on Principles of Knowledge Representation and Reasoning (KR)*, pages 57–67, 2006.
8. L. Kagal, T. Berners-Lee, D. Connolly, and D. Weitzner. Self-describing delegation networks for the Web. In *7th IEEE Int. Workshop on Policies for Distributed Systems and Networks (POLICY)*, pages 205–214. IEEE, 2006.
9. L. Kagal, T. Berners-Lee, D. Connolly, and D. J. Weitzner. Using Semantic Web technologies for policy management on the Web. In *21st National Conference on Artificial Intelligence (AAAI)*. AAAI Press, 2006.
10. G. Neumann and M. Strembeck. Design and implementation of a flexible RBAC-service in an object-oriented scripting language. In *8th ACM Conference on Computer and Communications Security (CCS)*, pages 58–67, 2001.
11. A. Rector. Representing specified values in OWL: "value partitions" and "value sets". Note NOTE-swbp-specified-values-20050517, W3C, May 2005.
12. R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-based access control models. *Computer*, 29(2):38–47, 1996.
13. T. Scavo and S. Cantor. Shibboleth Architecture, Technical Overview, Working Draft 02. Technical report, Internet2 Consortium, June 2005.
14. J. Seidenberg and A. L. Rector. Representing transitive propagation in OWL. In *25th International Conference on Conceptual Modeling (ER)*, pages 255–266, 2006.
15. M. Strembeck and G. Neumann. An integrated approach to engineer and enforce context constraints in RBAC environments. *ACM Trans. on Information and System Security*, 7(3):392–427, 2004.
16. W. Tolone, G.-J. Ahn, T. Pai, and S.-P. Hong. Access control in collaborative systems. *ACM Computing Surveys*, 37(1):29–41, 2005.
17. A. Toninelli, R. Montanari, L. Kagal, and O. Lassila. A semantic context-aware access control framework for secure collaborations in pervasive computing environments. In *5th International Semantic Web Conference*, pages 473–486, 2006.
18. A. Uszok, J. M. Bradshaw, M. Johnson, R. Jeffers, A. Tate, J. Dalton, and S. Aitken. KAoS policy management for semantic web services. *IEEE Intelligent Systems*, 19(4):32–41, 2004.
19. L. Wang, D. Wijesekera, and S. Jajodia. A logic-based framework for attribute based access control. In *ACM Workshop on Formal Methods in Security Engineering (FMSE)*, pages 45–55. ACM Press, 2004.
20. H. Xiao and I. F. Cruz. Integrating and Exchanging XML Data Using Ontologies. In *Journal on Data Semantics VI*, volume 4090 of *Lecture Notes in Computer Science*, pages 67–89. Springer, 2006.