Role Based Access Control and OWL

T. Finin[1], A. Joshi[1], L. Kagal[2], J. Niu[3], R. Sandhu[3], W. Winsborough[3], B. Thuraisingham[4]

[1] University of Maryland, Baltimore County
[2] Massachusetts Institute of Technology
[3] University of Texas at San Antonio
[4] University of Texas at Dallas

Abstract. Current access control research follows two parallel themes: many efforts focus on developing novel access control models meeting the policy needs of real world application domains while others are exploring new policy languages. This paper is motivated by the desire to develop a synergy between these themes facilitated by OWL. Our vision for the future is a world where advanced access control concepts are embodied in models that are supported by policy languages in a natural intuitive manner, while allowing for details beyond the models to be further specified in the policy language. In this paper we specifically study the relationship between the Web Ontology Language (OWL) and the Role Based Access Control (RBAC) model. Although OWL is a web ontology language and not specifically designed for expressing authorization policies, it has been used successfully for this purpose in previous work such as KAoS and Rei. We show two different ways to support the NIST Standard RBAC model in OWL and then discuss how the OWL constructions can be extended to model attribute-based RBAC or more generally attribute-based access control.

1 INTRODUCTION

There have been two prominent themes in access control research in recent years. One has focused on efforts to develop new access control models to meet the policy needs of real world application domains. These have led to several successful, and now well established, models such as the RBAC96 model [1], the NIST Standard RBAC model [2] and the RT model [3]. This line of research continues with recent innovations such as Usage Control models [4, 5]. In a parallel, and almost separate thread, researchers have developed policy languages for access control. These include industry standards such as XACML [6], but also academic efforts ranging from more practical implemented languages such as Ponder [7] to theoretical languages such as [8] and finally to Semantic Web based languages such as Rei [9] and KAoS [10]. Policy languages grounded in Semantic Web technologies allow policies to be described over heterogeneous domain data and promote common understanding among participants who might not use the same information model. This paper is motivated by the consideration that these two parallel efforts - access control models and Semantic Web based policy languages - need to develop synergy to enable the development of security infrastructures for emerging, open, and dynamic environments.

A policy language in the abstract without ties to a model gives the designer too much freedom and no guidance. Conversely a model may not have the machinery to express all the policy details of a given system or may deliberately leave important aspects unspecified. For instance the NIST Standard RBAC model only allows for the specific constraints of static and dynamic separation of duties - there is no room for any additional constraint. Our vision for the future is a world where mature access control concepts are embodied in models, which are supported by policy languages in a natural intuitive manner, while allowing for details beyond the models to be further specified in the policy language.

In this paper we specifically study the relationship between OWL and RBAC. OWL is a web ontology language and not specifically a language for authorization policies. Nonetheless it is not surprising that a powerful language such as OWL can support RBAC. Our motivation for using OWL is that it is a W3C standard that has been widely used for defining domain vocabularies, and has also been used previously to develop policy languages for the Web such as Rei and KAoS. Support for variations of RBAC in OWL can thus have immediate practical application.

2 ROWLBAC: RBAC IN OWL

Our goal is to define OWL ontologies that can be used to represent the RBAC security model and to show how they can be used to specify and implement access control systems. In doing so, we are able to identify which portion of RBAC can be modeled within OWL using Description Logic (DL), and which part requires other logical reasoning. In this section, we define two different approaches to modeling RBAC using OWL. For each approach, there is an ontology that defines the basic RBAC concepts including subjects, objects, roles, role assignments, and actions. Roles are central to RBAC and it is not surprising that much of the complexity in an RBAC system revolves around how roles are represented and managed. Our ontologies also define some special types of actions, including those for RBAC control such as activating a role, assigning a role, etc. and classes of actions to represent those that are permitted and those that are prohibited. As part of access control or monitoring, we need to recognize (or classify in DL) a specific action as being permitted, prohibited or (perhaps) fulfilling an obligation.

In addition to the basic RBAC ontology, each approach also has an ontology that models a specific domain ontology; defining the classes of roles, actions, subjects and objects in the domain, their relations and attributes as well as specifying which actions are permitted, prohibited or obligatory.

We use N3 syntax [5] for OWL in all our examples.

2.1 Scenario

We use a single scenario to illustrate our ROWLBAC approaches so that we can compare and contrast between them. We consider the case of "US persons" and permissions associated with them. The role hierarchy consists of two main classes: **USPerson** and **ForeignPerson**. **USPerson** is further divided into **Citizen**, **Resident**, and **Visitor** and Residents can be either **Permanent Residents**, **Permanent Residency Applicants**, or **Temporary Residents**. A *static separation of duty constraint* exists between **Resident** and **Citizen**, and **Permanent Resident** and **Temporary Resident**. Though a person may be both a **Visitor** and a **Temporary Resident**, he is not allowed to activate

[5] http://www.w3.org/TeamSubmission/2008/SUBM-n3-20080114/

both roles at once i.e. a *dynamic separation of duty constraint* exists between **Visitor** and **Resident**.

The instances we use are **Alice**, and **Bob**. **Alice**'s possible roles are **Citizen** and **Permanent Resident**, which should violate the *static separation of duty constraint* as **Permanent Resident** is a subclass of **Resident** and there is a SSOD constraint between **Resident** and **Citizen**. **Bob**'s can be a **Visitor**, and a **Temporary Resident**. **Alice** activates her **Citizen** role and now has the permission to **Vote**, **Work**, and perform **Jury duty**, which are associated with **Citizen** role. She then deactivates her **Citizen** role and activates the **Permanent Resident** role. She is still permitted to **Work** but can no longer **Vote** or perform **Jury duty**. Bob activates his **Visitor** role and finds that he is prohibited from **Working**. On activating his **TemporaryResident** role, he causes a *dynamic separation of duty violation*. He now tries activating the **Citizen** role but is not able to because it is not one of his possible roles.

2.2 Common Elements

The main concepts of RBAC including actions, subjects, and objects are common to both approaches of modeling RBAC with OWL.

Actions

An `Action` is a class that has exactly one subject, which must be an instance of the `Subject` class, and one object or resource, which must be an instance of the `Object` class.

```
Action a rdfs:Class;
  rdfs:subClassOf [ a owl:Restriction;
                    owl:onProperty subject;
                    owl:cardinality 1 ] .
subject a rdfs:Property, owl:FunctionalProperty;
  rdfs:domain Action;
  rdfs:range Subject.
object a rdfs:Property, owl:FunctionalProperty;
  rdfs:domain Action
  rdfs:range Object.
```

This can be easily modified to make the object optional to describe actions that do not have an object (e.g., login) and to have additional properties for time, location, manner, instrument, etc. To control access, we introduce two important `Action` subclasses for permitted and prohibited actions: `PermittedAction` and `ProhibitedAction`. Every action is either permitted or prohibited and no action can be both permitted and prohibited. We can express this in our ontology as:

```
PermittedAction rdfs:subClassOf Action;
   owl:disjointWith ProhibitedAction.

ProhibitedAction rdfs:subClassOf Action;
   owl:disjointWith PermittedAction.

Action
   owl:equivalentClass
   [ a owl:Class;
       owl:unionOf (PermittedAction ProhibitedAction) ].
```

Subjects and Objects

The `Subject` class represents things that can serve as a subject of an action. The RBAC ontology defines some key properties that a subject can have (depending on the details of the representation) and leaves the specification of additional properties and subclasses to the specific domain model. The `Object` class represents things that can be the object of an RBAC action and is basically defined as a class and can be given additional properties, if required by the domain.

2.3 Approach 1: Roles as Classes

A natural way to represent RBAC roles in OWL is as classes to which individual subjects can belong. We represent role hierarchies by OWL class hierarchies in which the inheritance relation is the inverse of the role dominance relations, meaning that the ordering is reversed: a role represented by a subclass dominates a role represented by its superclass. This corresponds to the intuition that in role hierarchies, members get more privilages as one moves up the hierarchy, while in class hierarchies, classes get more attributes as you move down. Note that OWL supports multiple inheritance.

Suppose we want to model the US Persons hierarchy; we will have three base classes, **Citizen**, **Resident**, and **Visitor**, which are defined as subclasses of a `Role` class. The other classes in the domain are defined as subclasses of one of these classes. We have an active role, which is an `ActiveRole`, associated with each role class in the ontology via the `activeForm` property. OWL classes represents sets of individuals, so the Citizen class is the set of individuals who have the **Citizen** role as one of their possible roles and the **ActiveCitizen** role is the set of individuals who have activated their **Citizen** role. Since a subject can activate a role only if it is one of her possible roles, each active role class is a sub-class of its associated role class. In a flat RBAC system we can define a class and active role class for each possible role without defining subclass relationships between them.

```
rbac:Role a owl:Class.
rbac:ActiveRole a owl:Class.
```

The **US Person** role class would be represented as

```
USPerson rdfs:subClassOf rbac:Role.
ActiveUSPerson rdfs:subClassOf rbac:ActiveRole,
    rdfs:subClassOf USPerson.
USPerson rbac:activeForm ActiveUSPerson.
```

If **Alice** is in the **Citizen** role and has activated it, and Bob is in **Visitor** and **TemporaryResident** role, we would assert

```
Alice a Citizen, ActiveCitizen.
Bob a Visitor, TemporaryResident.
```

Hierarchical roles.

Using OWL classes to represent RBAC roles makes adding hierarchical roles easy. We can use `rdfs:subclassOf` to define sub-roles. If we want **Citizen**, **Resident**, and **Visitor** roles to be sub-roles of **US Person**, and **Permanent Resident** and **Temporary Resident** to be sub-roles of **Resident**, we need add the following assertions

```
Citizen rdfs:subclassOf USPerson.
Resident rdfs:subclassOf USPerson.
Visitor rdfs:subClassOf USPerson.

PermanentResident rdfs:subclassOf Resident.
TemporaryResident rdfs:subclassOf Resident.
```

Static separation of duty

An RBAC static separation of duty constraint specifies pairs of roles where any subject can only have one of the pair as a possible role. We might, for example, specify that no one have access to both the **Citizen** and **Resident** role. We can specify this constraint in our OWL representation by asserting that the two classes that represent them are disjoint. We use an existing OWL property, **disjointWith**, for this purpose

```
Citizen owl:disjointWith Resident.
```

Dynamic separation of duty

An RBAC dynamic separation of duty constraint holds between two roles when no subject can have both simultaneously active. Again, we can use OWL's **disjointWith** property to specify that this constraint holds but this time between the active roles associated with the classes. If we want a dynamic separation of duty constraint to hold between the **Visitor** and **Temporary Resident** roles, we need to assert

```
ActiveVisitor owl:disjointWith ActiveTemporaryResident
```

Associating Permissions with Roles

In order to associate permissions, or prohibitions with roles, we use OWL class expressions [6] to create classes of permitted or prohibited actions. As only **Citizens** are allowed to vote, we create an action, **PermittedVoteAction**, which is the subclass of **rbac:PermittedAction** and whose subjects can only be individuals who have activated their **Citizen** role.

```
PermittedVoteAction a rdfs:Class;
   rdfs:subClassOf rbac:PermittedAction;
   owl:equivalentClass [
      a owl:Class;
      owl:intersectionOf
        (  Vote
             [ a owl:Restriction;
                 owl:allValuesFrom ex:ActiveCitizen;
                 owl:onProperty rbac:subject
             ]
         )
    ] .
```

Enforcing RBAC

In this approach, we exploit the ability of DL to easily model classes and use OWL constructs to represent roles, subjects, actions, and to associate permissions/prohibitions with roles. We use DL subsumption reasoning to figure out whether users are permitted to perform actions associated with roles. However, for enforcing *static separation of duty* and *dynamic separation of duty* constraints, and for role activation and deactivation we use rules in N3Logic, which is a rule language that allows rules to be expressed in a Web environment using RDF [11]. Other rule languages that support OWL could also have been used instead. Please refer to Table 1 for examples of some rules used.

Dynamic separation of constraints	Role Activation

`{ ?ACTION a ActivateRole;`
`  subject ?SUBJ;`
`  object ?RNEW.`
`?RNEW activeForm ?ARNEW.`
`?S a ?RCURRENT.`
`?RCURRENT activeForm ?ARCURRENT.`
`?ARNEW owl:disjointWith ?ARCURRENT.`
`} => { ?ACTION a ProhibitedRoleActivation; subject ?SUBJ;`
`object ?RNEW; role ?RCURRENT;`
`justification "Violates DSOD constraint".}.`
	`{ ?ACTION a ActivateRole;`
	`subject ?SUBJ;`
	`object ?ROLE.`
	`?SUBJ a ?ROLE.`
	`?ROLE activeForm ?AROLE.`
	`?AROLE a ActiveRole.`
	`} => { ?ACTION a PermittedRoleActivation;`
	`subject ?SUBJ; object ?ROLE.`
	`?SUBJ a ?AROLE }.`

Table 1. Enforcing RBAC in Approach 1

[6] OWL Class Expressions: `http://www.w3.org/TR/2004/REC-owl-guide-20040210/#ComplexClasses`

2.4 Approach 2: Roles as Values

An alternate way to model roles is as instances of the generic `Role` class using two properties `role` and `activeRole` to link a subject to her possible and active roles, respectively. This representation is on the surface simpler than the previous one, but it requires special rules to implement hierarchical roles. We define the `role` and `activeRole` properties as follows

```
rbac:Role a owl:Class.
rbac:role a owl:ObjectProperty;
    rdfs:domain rbac:Subject;
    rdfs:range rbac:Role.
rbac:activeRole rdfs:subPropertyOf rbac:role.
```

Note that the activeRole property is a sub-property of role, since a subject's activated roles must be a subset of her possible roles. Since it is a sub-property, it inherits the domain and range of the role property. For our running example, we define flat roles as instances of Role.

```
USPerson a rbac:Role.
Citizen a rbac:Role.
Resident a rbac:Role.
```

If **Alice** is in the **Citizen** role and has activated it, and Bob is in **Visitor** and **TemporaryResident** role, we would assert

```
Alice rbac:role Citizen;
    rbac:activeRole Citizen.
Bob rbac:role Visitor, TemporaryResident.
```

Hierarchical roles

Adding the capability to define role hierarchies is more difficult in this representation and requires adding rules to the ontology, either in SWRL [12] or N3 [11], depending on the kind of reasoner used. We start by defining a property, `subRole`, which holds between two roles to state that one is the sub-role of the other. We define `subRole` as

```
rbac:subRole a owl:TransitiveProperty;
  rdfs:domain rbac:Role;
  rdfs:range rbac:Role.
```

We can then use `subRole` to specify relationships between roles to create the role hierarchy. For example, portion of the scenario domain can be defined as

```
Citizen rbac:subRole USPerson.
Resident rbac:subRole USPerson.
Visitor rbac:subRole USPerson.

PermanentResident rbac:subRole Resident.
TemporaryResident rbac:subRole Resident.
```

Static and dynamic separation of duty

The representation of *static and dynamic separation of duty constraints* is also more complicated here than in the earlier 'roles as classes' approach. It requires the introduction of properties to link the constrained roles. We define two properties: `ssod` to represent *static separation of duty constraints* and `dsod` for *dynamic separation of duty constraints* properties. These properties hold between role instances and are defined to be symmetric and transitive. The `ssod` property is defined as

```
rbac:ssod a owl:SymmetricProperty, owl:TransitiveProperty;
    rdfs:domain rbac:Role;
    rdfs:range rbac:Role.
```

For example, to specify a *static separation of duty constraint* between roles **Resident** and **Citizen** and a dynamic separation of duty constraint between **Visitor** and **TemporaryResident**, we would assert the following.

```
Resident rbac:ssod Citizen.
Visitor rbac:dsod TemporaryResident.
```

Associating Permissions with Roles

As roles in the domain are instances of `Role` class, they can be directly associated with action that are permitted or prohibited for individuals in that role. We introduce two properties namely `permitted` and `prohibited` for this purpose.

```
rbac:permitted a rdfs:Property;
  rdfs:domain Role;
  rdfs:range Action.

rbac:prohibited a rdfs:Property;
  rdfs:domain Role;
  rdfs:range Action.
```

Consider the permitted actions, **Vote**, **Work**, and **Jury Duty**, associated with the **Citizen** role

```
Citizen rbac:permitted Vote, Work, JuryDuty.
```

Prohibitions can be described similarly

```
Visitor rbac:prohibited Work.
```

Enforcing RBAC

Though this approach leads to a more concise RBAC specification, we are unable to utilize DL reasoning for most of the reasoning including role hierarchy reasoning, role activation, separation of duty constraints, and permission/prohibition association. We need to introduce rules to do each of this. As before, we have developed rules in N3Logic. Please refer to Table 2 for examples of some rules used.

Role hierarchies

```
# role inheritance.
{ ?S role ?R.
  ?R subRole ?R2
} => {?S role ?R2.}.

# activerole inheritance.
{ ?S activeRole ?R.
  ?R subRole ?R2
} => {?S activeRole ?R2.}.
```

Action permitted

```
{ ?A a ?RACTION; subject ?S.
  ?RACTION a Action.
  ?ROLE permitted ?RACTION.
  ?S activeRole ?ROLE.
} => { ?A a PermittedAction;
          role ?ROLE;
          action ?RACTION; subject ?S }.
```

Table 2. Enforcing RBAC in Approach 2

2.5 Comparing the two approaches

An advantage of defining roles as classes is that queries about a particular access request (*Can John use printer p43?*) and queries about a general class of access requests (*Can every student use lab printers?*) can be answered efficiently using a standard DL reasoner through subsumption reasoning. We say that description A subsumes description B when A logically entails B. Thus, *professor using a printer* subsumes *assistant professor using a color printer* which might in turn subsume *John using printer p43*. Given a description, either of an instance or a class, a DL reasoner can efficiently find all of the other descriptions that it subsumes and that are subsumed by it.

If we treat roles as values the specification is simpler and more concise but can not exploit a DL reasoner's ability to determine the subsumption relationships between a query and all of the classes in our policy. We can, of course, still take a description of an instance action (e.g., *John using printer p43*) and classify it as either permitted or prohibited. What we can not do, is determine if a description representing a generalized action is necessarily permitted or prohibited.

Both these approaches, however, have a fundamental problem with managing state changes due to the essentially monotonic nature of RDF/OWL [13]. This implies non-monotonic state changes such as role deactivations, and modifying

role-permission assignments must be handled outside the reasoners. Once the changes have been applied, the reasoners can be used for queries within the context of the current state.

3 DISCUSSION: BEYOND RBAC

Our motivation is not just to model RBAC concepts in OWL, but to develop a foundation on which we can build newer ideas for information assurance, including attribute based access control and usage control. This section identifies some issues that go beyond RBAC, some of the challenges for modeling them in OWL, and possible approaches to accommodating them.

3.1 Attribute based access control

Representing access constraints based on general attributes of an action, including constraints on its subject and object, follows naturally from our approach. This provides direct support to a more general model of attribute-based access control [14] which can be used even when the principals are unknown, assuming that their attributes can be reliably determined.

For example, suppose we want to specify a policy constraint *faculty can use any printer located in a classroom*. To do this we would first extend the domain model to include a *Place* class to represent physical spaces with subclasses for various subtypes (e.g., *Office*, *Classroom*, *Lab*) and a *location* property that links an *Object* to a *Place*. Our constraint can then be easily expressed as a new class of permitted action using a restriction or by a rule in N3 or SWRL. Here is how it might be expressed as a rule in N3.

```
{ ?A a rbac:Action;
       rbac:subject ?S;
       rbac:object ?O.
  ?S a Faculty.
  ?O a Printer; location ?L.
  ?L a Classroom
} => { ?A a rbac:PermittedAction }.
```

The same constraint can easily be encoded in description logic without resort to the rule sublanguage and correctly handled by a standard description logic reasoner. We show the N3 rule form for clarity. More complicated cases might involve roles and constraints on both the action's subject and object. For example, we could specify that *A university member can use any device that is located in her office*.

```
{ ?A a rbac:Action;
       rbac:subject ?S;
       rbac:object ?O.
  ?S a UniversityPerson; office ?L,
  ?O a Device; location ?L.
} => { ?A a rbac:PermittedAction }.
```

While this looks simple when expressed in a rule format, the constraint that the value of the object's location and subject's office represents (in description logic terms) a *role value map*, the inclusion which in a description logic system is known to make computing subsumption undecidable [15], in general. However, with suitable restrictions (e.g., to a Boolean combination of basic roles), the use of role value maps does not affect decidability or worst-case reasoning complexity [16].

Note that a description logic reasoner's ability to compute subsumption can be used to provide general answers to a question like "What devices can Marie use" by generating descriptions from the subsuming policy classes.

type	syntax	description	OWL encoding
1	A.r ← D	simple member	D a A-r
2	A.r ← B.r1	simple inclusion	B-r rdfs:subClassOf A-r
3	A.r ← B.r1.r2	linking inclusion	problematic
4	A.r ← B.r1 ∧ C.r2	intersection inclusion	[owl:intersectionOf (B-r1 C-r2)] rdfs:subClassOf A-r

Fig. 1. RT has four types of policy rules. RT rules of types 1, 2 and 4 can be easily encoded in OWL. Type 3 rules are problematic.

As a member of the faculty, Marie can use any printer located in a classroom. As a university member, she can use any devices located in her office.

3.2 Security analysis

An administrative policy specifies and constrains who can make what kinds of changes to a policy. Exploring the consequences of an administrative policy involves reasoning about possible changes in a fundamental way. Given an administrative policy and an initial object policy, one might want to know whether it is possible for the access control policy to evolve in such a way that some individual comes to have simultaneous access to targets X and Y or whether every subject in some role will always have access to a given target [17]. In general, to answer such queries requires us to consider all the possible changes to a policy, both adding and subtracting roles and privileges, that might take place.

While some queries about the consequences of an administrative policy can be handled by current OWL reasoners, others can not. We will only give an example of a kind of constraint that can be modeled in OWL and an example of one that can not. Our examples will use the RT role-based policy language [18] designed to support highly decentralized attribute-based access control.

The RT has four types of statements shown in Figure 1. Type 1 statements introduce individual principals to roles. For example, *Alice.friend* ← *Bob* identifies Bob as a friend of Alice. Type 2 statements provide a form of delegation via the implication that principals in one role are necessarily in another. For example, the statement *Alice.friend* ← *Bob.friend* specifies that if a principal is a friend of Bob, then they are also a friend of Alice. Type 3 statements allow one to delegate to all members of a role. For example, the statement *Alice.friend* ← *Bob.friend.friend* says that any friend of Bob's friends is also a friend of Alice. Type 4 statements introduce intersection – a principal must be in two roles in order to be included. For example, *Alice.friend* ← *Bob.friend* ∧ *Carl.friend* states that only principals who are both Bob's friends and Carl's friends are in the set of Alice's friends.

We can easily represent the RT roles as OWL classes and principals as instances. Since N3's syntax won't allow us to 'dot' in a class name, we use A-r instead of A.r to denote A's r role. Figure 1 shows how the different RT statements are encoded, assuming A-r, B-r and C-r are defined as owl:Class.

The type 3 roles do not have a clean representation in OWL. Modeling these requires descriptions that involve "role chains" also known as "role composition" in the description logic literature. Unrestricted role composition can introduce undecidability and this feature is not included in the current OWL standard, although a restricted form is included in a proposed OWL 1.1 standard. We can, of course, model such role chain constraints as rules, but current OWL reasoners will not guarantee complete reasoning in all cases. We believe, however, that the

HQ.marketing ← HR.managers
HQ.marketing ← HQ.staff
HQ.marketing ← HR.sales
HQ.marketing ← HQ.marketingDelg ∧ HR.employee
HQ.ops ← HR.managers
HQ.ops ← HR.manufacturing
HQ.marketingDelg ← HR.managers.access
HR.employee ← HR.managers
HR.employee ← HR.sales
HR.employee ← HR.manufacturing
HR.employee ← HR.researchDev
HQ.staff ← HR.managers
HQ.staff ← HQ.specialPanel ∧ HR.researchDev
HR.manager ← Alice
HR.researchDev ← Bob

Growth and shrink restricted roles: HQ.marketing, HQ.ops HR.employee, HQ.marketingDelg, HQ.staff

Fig. 2. Example RT access policy with growth and shrink restricted roles.

use of role composition in RT can be handled by a DL reasoner. A more serious problem arises, however, when one considers reasoning about the consequences of policy changes. Given OWL's foundation in classical first order logic, it works well when modeling positive changes (i.e., additions of sentences) but not when modeling negative ones (i.e., retraction of sentences).

A given policy state evolves into another as principals issue and revoke policy statements. We want to analyze whether security properties under the assumption that some of roles are under our control or otherwise trusted, but others are not. This can be modeled [17] as two types of roles used to determine the reachable policy states – growth-restricted and shrink-restricted. Growth-restricted roles will not have new statements defining them added and shrink-restricted roles will not have statements defining them removed. These restrictions are not actually enforced, but are assumptions underlying the analysis. Their presence enables the analysis to provide us with reassurances of constraints like, "So long as the people I trust don't change the policy without first running the analysis, only company employees will be able to access the secret database."

Representing a shrink restricted description is trivial, since OWL is based on a monotonic logic. All OWL descriptions are shrink restricted. On the other hand, roles that are neither shrink restricted nor growth restricted can be handled by simply dropping all RT statements defining them. Unfortunately, representing growth restricted roles that are not also shrink restricted is somewhat problematic. This is because, given a specification, partial or complete, of a class, it is not possible in OWL's framework to retract parts of the specification. If we assume that a role is shrink restricted, it is possible to model it as either growth enabled or growth restricted. A growth enabled role is easy since that is the default case for OWL descriptions. OWL assumes an "open world" semantics in which it is always possible to add more knowledge, so by default, descriptions are assumed to be partial. If we want to model a role as being "growth restricted", we can do so by making its OWL description be both necessary and sufficient. (This corresponds to the Clarke completion.)

Consider the access control policy of a company that has a marketing strategy and an operations plan that it must protect from competitors, while being accessible to those employees with a need to know. A policy in RT is shown in Figure 2. Examples of properties to check include the following.

- **Restriction.** Are the marketing strategy and operations plan only available to employees? The property holds if (HR.employee $\supseteq$ HQ.marketing $\wedge$ HR.employee $\supseteq$ HQ.ops). TRUE
- **Access.** Does everyone who has access to the operations plan also have access to the marketing plan? The is true if (HQ.marketing $\supseteq$ HQ.ops) TRUE
- **Availability.** Will Alice always have access to the marketing plan? This is true if (Alice $\in$ HQ.marketing). FALSE.
- **Safety.** Will anyone other than Alice and Bob ever be able to access the marketing plan? This will be true if it follows that (x $\neg \in$ HQ.marketing) for a Skolem individual x. FALSE

The first two can be proven true by Pellet since they involve classes that are shrink restricted. The second two can not be proven since they involve roles that are not shrink restricted. Alternative techniques such as model checking [19] are needed to fully evaluate some of these properties.

4 RELATED WORK

There have been some recent efforts to look at OWL as a represenation language for RBAC policies. Di et al [20] suggest modeling Roles, Users, Permissions, and Session as classes, with properties to relate users to roles and roles to permission(s). There are also functional mappings between sessions and roles (i.e. the active role for the session) user to session. However, while the authors do not make this explicit, they need to step outside of OWL and add rules to specify separation of duty and prerequisite constraints. This means that the efficient DL reasoners will not be able to deal with policies specified using their approach. It is also unclear if this approach can handle queries that deal with classes not instances, e.g. *"Is there a faculty member authorized to change grades?"*).

A natural way to represent role hierarchies in OWL is using subclass axioms such as in [21, 22]. Kolovski's approach [21] is similar to our first approach including the use of **owl:disjointWith** to specify *static separation of duty constraints*, however, they do not consider other RBAC concepts such as *active roles*, *role activation*, and enforcing *dynamic separation of duty constraints*. Heilili et al [22] also define users and roles as classes. However, in order to handle negative authorizations (which is an extension of RBAC) each role has two correspoding classes, each permission or prohibition on a resource has corresponding classes for roles and users. In other words, for each permission, we have a class of roles that have that permission, and then a class of users who have that permission. Similarly for each prohibition.

5 CONCLUSION

In an attempt to harmonize formal access control models and declarative policy languages, we studied the relationship between the RBAC security model and OWL and represented the RBAC model in OWL. We believe that this will help in developing security frameworks with well understood and verifiable security properties for open, dynamic environments, which require coordination across multiple organizations and integration of different data formats. In this paper, we described two possible approaches to RBAC in OWL, representing roles as classes and sub-classes in one approach and as attributes in an alternate approach. We hope to use these OWL models as a starting point for building new ideas about information assurance and propose to model and reason over general attribute based access control such as the UCON model in a similar manner.

REFERENCES

1. Sandhu, R., Coyne, E.J., Feinstein, H.L., Youman, C.E.: Role-based access control models. IEEE Computer **29**(2) (February 1996) 38–47
2. Ferraiolo, D., Sandhu, R., Gavrila, S., Kuhn, D., Chandramouli, R.: Proposed NIST standard for role-based access control. ACM Transactions on Information and System Security (TISSEC) **4**(3) (2001) 224–274
3. Li, N., Mitchell, J., Winsborough, W.: Design of a role-based trust-management framework. Security and Privacy, 2002. Proceedings. 2002 IEEE Symposium on (2002) 114–130
4. Park, J., Sandhu, R.: The UCON$_{ABC}$ usage control model. ACM Transactions on Information and System Security **5**(6) (2007)
5. Pretschner, A., Hilty, M., Basin, D.: Distributed usage control. Communications of the ACM **49**(9) (2006) 39–44
6. Moses, T., et al.: eXtensible Access Control Markup Language (XACML) Version 2.0. OASIS Standard **200502** (2005)
7. Damianou, N., Dulay, N., Lupu, E., Sloman, M.: The ponder policy specification language. Lecture Notes in Computer Science **1995** (2001)
8. Jajodia, S., Samarati, P., Subrahmanian, V.: A Logical Language for Expressing Authorizations. Proceedings of the 1997 IEEE Symposium on Security and Privacy (1997) 31
9. Kagal, L., Finin, T., Joshi, A.: A policy language for pervasive systems. In: Fourth IEEE International Workshop on Policies for Distributed Systems and Networks. (2003)
10. Tonti, G., Bradshaw, J.M., Jeffers, R., Montanar, R., Suri1, N., Uszok1, A.: Semantic web languages for policy representation and reasoning: A comparison of kaos, rei, and ponder. In: Proceedings of the 2nd International Semantic Web Conference (ISWC2003), Springer-Verlag (2003)
11. Berners-Lee, T., Connolly, D., Kagal, L., Hendler, J., Schraf, Y.: N3Logic: A Logical Framework for the World Wide Web. Journal of Theory and Practice of Logic Programming (TPLP), Special Issue on Logic Programming and the Web (2008)
12. Horrocks, I., Patel-Schneider, P., Boley, H., Tabet, S., Grosof, B., Dean, M.: SWRL: A semantic web rule language combining OWL and RuleML. W3C Member Submission **21** (2004)
13. Hayes, P., McBride, B.: RDF Semantics. `http://www.w3.org/TR/rdf-mt/` (2004)
14. Wang, L., Wijesekera, D., Jajodia, S.: A logic-based framework for attribute based access control. Proceedings of the 2004 ACM workshop on Formal methods in security engineering (2004) 45–55
15. Schmidt-Schauss, M.: Subsumption in KL-one is undecidable. Fachber. Informatik, Univ (1988)
16. Baader, F.: Restricted role-value-maps in a description logic with existential restrictions and terminological cycles. Proc. DL 2003
17. Li, N., Mitchell, J., Winsborough, W.: Beyond proof-of-compliance: security analysis in trust management. Journal of the ACM (JACM) **52**(3) (2005) 474–514
18. Li, N., Mitchell, J.: RT: A Role-based Trust-management Framework. DARPA Information Survivability Conference and Exposition (DISCEX) 123–139
19. Clarke, E.M., Grumberg, O., Peled, D.A.: Model Checking. The MIT Press, Cambridge, Massachusetts, USA (1999)
20. Di, W., Jian, L., Yabo, D., Miaoliang, Z.: Using semantic web technologies to specify constraints of rbac. Parallel and Distributed Computing, Applications and Technologies, 2005. PDCAT 2005. Sixth International Conference on (05-08 Dec. 2005) 543–545
21. Kolovski, V., Hendler, J., Parsia, B.: Analyzing web access control policies. In: WWW '07: Proceedings of the 16th international conference on World Wide Web, New York, NY, USA, ACM (2007) 677–686
22. Heilili, N., Chen, Y., Zhao, C., Luo, Z., Lin, Z.: An owl based approach for rbac with negative authorization. Lecture Notes in Computer Science **4092** (2006) 164