R*OWL*BAC - Representing Role Based Access Control in *OWL*

T. Finin, A. Joshi
Univ. of Maryland, Baltimore County
finin,joshi@cs.umbc.edu

L. Kagal
Massachusetts Institute of Technology
lkagal@csail.mit.edu

J. Niu, R. Sandhu, W. Winsborough
Univ. of Texas at San Antonio
niu,winsboro@cs.utsa.edu,
ravi.sandhu@utsa.edu,

B. Thuraisingham
Univ. of Texas Dallas
bhavani.thuraisingham@utdallas.edu

ABSTRACT

There have been two parallel themes in access control research in recent years. On the one hand there are efforts to develop new access control models to meet the policy needs of real world application domains. In parallel, and almost separately, researchers have developed policy languages for access control. This paper is motivated by the consideration that these two parallel efforts need to develop synergy. A policy language in the abstract without ties to a model gives the designer little 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. 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.

This paper studies 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. OWL is a leading specification language for the Semantic Web, making it a natural vehicle for providing access control in that context. In this paper 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. We further examine and assess OWL's suitability for two other access control problems: supporting attribute based access control and performing security analysis in a trust-management framework.

SACMAT'08, June 11–13, 2008, Estes Park, Colorado, USA.

Categories and Subject Descriptors

K.6.5 [**Management of Computing and Information Systems**]: Security and Protection; D.4.6 [**Operating Systems**]: Security and Protection—*Access Controls*

General Terms

Security, Theory, Languages, Design

Keywords

RBAC, Access Control Model, OWL, Ontology, Semantic Web

1. INTRODUCTION

As applications become more sophisticated, intelligent, and function in open and dynamic environments, they require greater degrees of decision making and autonomy. A long range vision is to have societies of intelligent, adaptive, autonomous agents, but even today, we find the new levels of autonomy emerging in infrastructures such as Grid computing, web services and pervasive computing. These systems must exchange information about services offered and sought and negotiate for information sharing. A new challenge is to secure these applications in open, dynamic environments.

Two parallel themes in access control research are prominent 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 [28], the NIST Standard RBAC model [9] and the RT model [20]. This line of research continues with recent innovations such as Usage Control models [24, 25]. In a parallel, and almost separate thread, researchers have developed policy languages for access control. These include industry standards such as XACML [23], but also academic efforts ranging from more practical implemented languages such as Ponder [7] to theoretical languages such as [15] and finally to Semantic Web based languages such as Rei [16] and KAoS [34]. 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—accesss control models and Semantic Web based policy languages—need to develop synergy to enable the development of security infrastructures with verifiable security properties 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. On the administrative front, the NIST Standard RBAC model is silent on how users and permissions are added to roles. On both counts a policy language built around the NIST Standard RBAC model would be useful in specifying these additional but very important details that are not captured directly in the model. 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. We expect a many-to-many relation between access control models and policy languages in that a single model such as the NIST Standard can be supported by multiple policy languages, and conversely a single policy language such as Ponder can support multiple models.

Now what does it mean for a policy language to support a model? Clearly there must be some way of expressing the components and behavior of the model in the language. To be useful this expression must be "natural" in some intuitive sense so that it provides leverage to a human in thinking along the lines of the model while expressing model details in the language. As we will see an expressive language will have multiple ways of supporting a model, each with their own pros and cons. In the context of this paper, OWL [22] can represent roles as classes and sub-classes in one approach and as attributes in an alternate approach. None of this should be surprising to computer scientists.

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. The motivation for picking RBAC is due to its real world success and considerable academic study. Support for variations of RBAC in OWL can thus have immediate practical application. Being interested in OWL's general suitability for use with access control problems, we also briefly explore supporting attribute based access control and performing security analysis in a trust-management framework by using OWL[1].

The paper is organized as follows. First we show two different ways to support the NIST Standard RBAC model in OWL as discussed above. Then we show how the OWL constructions can be extended to model attribute-based RBAC [1] or more generally attribute-based access control, and we assess OWL's ability to support security analysis. We conclude the paper with a few suggestions for future work.

[1]In future work we would like to extend this study to the relationship between OWL and more elaborate models such as usage control, as well as extend OWL with additional deontic policy elements such as in Rei. However, this is beyond the current scope.

2. SEMANTIC WEB AND OWL

The Semantic Web refers to both a vision and a set of technologies. The vision was first articulated by Tim Berners-Lee as an extension to the existing web in which knowledge and data could be published in a form easy for computers to understand and reason with. Doing so would support more sophisticated software systems that share knowledge, information and data on the Web just as people do by publishing text and multimedia. Under the stewardship of the W3C, a set of languages, protocols and technologies have been developed to partially realize this vision, to enable exploration and experimentation and to support the evolution of the concepts and technology.

The current set of W3C standards are based on RDF [17], a language that provides a basic capability of specifying graphs with a simple interpretation as a "semantic network" and serializing them in XML and several other popular Web systems (e.g., JSON). Since it is a graph-based representation, RDF data are often reduced to a set of *triples* where each represents an edge in the graph (*Person32 hasMother Person45*) or altrnatively, a binary predication (e.g., *hasMother(Person32,Person45)*. The Web Ontology Language OWL [5] is a family of knowledge representation languages based on Description Logic (DL) [3] with a representation in RDF. OWL supports the specification and use of ontologies that consist of terms representing individuals, classes of individuals, properties, and axioms that assert constraints over them. The axioms can be realized as simple assertions (e.g., *Woman is a sub-class of Person*, *hasMother is a property from Person to Woman*, *Woman and Man are disjoint*) and also as simple rules.

The use of OWL to define policies has several very important advantages that become critical in distributed environments involving coordination across multiple organizations. First, most policy languages define constraints over classes of targets, objects, actions and other constraints (e.g., time or location). A substantial part of the development of a policy is often devoted to the precise specification of these classes, e.g., the definition of what counts as a *full time student* or a *public printer*. This is especially important if the policy is shared between multiple organizations that must adhere to or enforce the policy even though they have their own native schemas or data models for the domain in question. The second advantage is that OWL's grounding in logic facilitates the translation of policies expressed in OWL to other formalisms, either for analysis or for execution.

3. 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 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. One aspect is the kind of RBAC system we want to model. Common variations include the following [10]:

- Flat RBAC: users get permissions through roles, many-to-many user-role assignment, many-to-many permission-role assignment, users can use permissions of multiple roles simultaneously.
- Hierarchical RBAC: Flat RBAC + must support role hierarchy
- Constrained RBAC: Hierarchical RBAC + must support static and dynamic separation of duties (SOD)
- Symmetric RBAC: Constrained RBAC + must support permission-role review

Our two approaches mainly differ in their representation of *roles* but they support all combinations of the above features. 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. For example, if we are writing a policy to control use of devices in a wireless environment, we will need to define appropriate roles (e.g., admin, powerUser, user, guest), objects (e.g., printer, videoRecorder, display), and subjects.

To use the system, we will also need some data about instances (e.g., Mary, Printer13) in the domain and some use cases to test the design where subjects take on various roles and attempt to perform actions.

We use the N3 representation syntax[2] for OWL in all our examples.

3.1 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**. Please refer to Figure 1 for details of this role hierarchy. 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 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 (meaning parent role) 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 **Work**ing. 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]http://www.w3.org/TeamSubmission/2008/SUBM-n3-20080114/

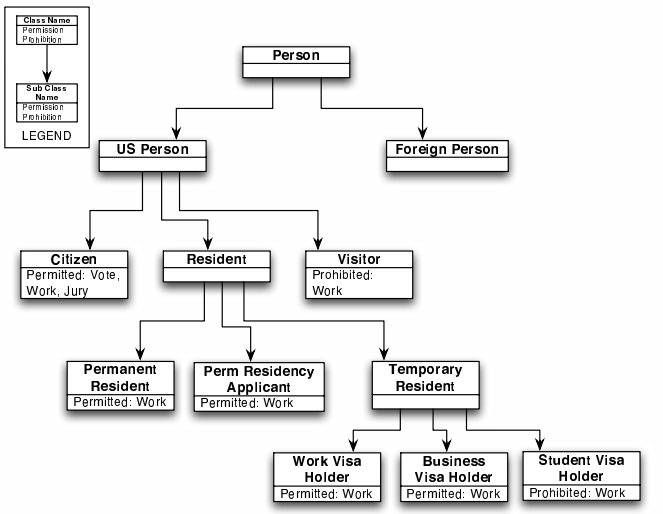

Figure 1: US Persons role hierarhcy

3.2 Common Elements

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

3.2.1 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.
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 to 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) ].
```

3.2.2 Subjects

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.

```
Subject a rdfs:Class.
```

3.2.3 Objects

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.

```
Object a rdfs:Class.
```

3.3 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. (In the following examples, `rbac:X` refers to the term X as defined within the rbac namespace) `Role` and `ActiveRole` are defined in our ontology as

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

In order to model roles using this approach, each role is defined as follows

```
<RoleName> rdfs:subclassOf rbac:Role.
<ActiveRoleName> rdfs:subclassOf rbac:ActiveRole;
                 rdfs:subclassOf <RoleName>.
<RoleName> rbac:activeForm <ActiveRoleName>.
```

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.
```

3.3.1 Hierarchical roles.

Using OWL classes to represent RBAC roles makes adding hierarchical roles easy. We can use `rdfs:subclassOf` to define sub-roles such as

```
<RoleName> rdfs:subclassOf <SuperRoleName>.
```

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.
```

3.3.2 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.
```

3.3.3 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.
```

3.3.4 Associating Permissions with Roles

In order to associate permissions, or prohibitions with roles, we use OWL class expressions [3] 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.

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

```
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
             ]
         )
    ] .
```

3.3.5 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 [6]. Other rule languages that support OWL could also have been used instead.

For enforcing *dynamic separation of constraints* we use the following rule

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

The role activation rule is

```
{ ?ACTION a ActivateRole;
       subject ?SUBJ;
       object ?ROLE.
  ?SUBJ a ?ROLE.
  ?ROLE activeForm ?AROLE.
  ?AROLE rdfs:subClassOf ActiveRole.
} => { ?ACTION a PermittedRoleActivation;
            subject ?SUBJ; object ?ROLE.
            ?SUBJ a ?AROLE }.
```

3.4 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.
```

3.4.1 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 [14] or N3 [6], 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 sub role relationships between roles and 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.
```

In order to model the roles using this approach, each role is defined as follows

```
<RoleName> a rbac:Role.
<RoleName> rbac:subRole <SuperRoleName>.
```

3.4.2 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.

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

rbac:dsod 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.
```

3.4.3 Associating Permissions with Roles

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

Property	Roles as Classes	Roles as Values
Defining Roles	<RoleName> rdfs:subclassOf rbac:Role. <ActiveRoleName> rdfs:subClassOf rbac:ActiveRole. <ActiveRoleName> rdfs:subclassOf <RoleName>. <RoleName> rbac:activeForm <ActiveRoleName>	<RoleName> a rbac:Role.
Role Hierarchy	<RoleName> rdfs:subclassOf <SuperRoleName>	<RoleName> rbac:subRole <SuperRoleName>
Permission Association	OWL class expression	<RoleName> rbac:permitted <Action>
Static Separation of Duty Constraint	<Role1> owl:disjointFrom <Role2>	<Role1> rbac:ssod <Role2>
Dynamic Separation of Duty Constraint	<ActiveRole1> owl:disjointFrom <ActiveRole2>	<Role1> rbac:dsod <Role2>
Queries	role activation permitted, separation of duty, access monitoring	role activation permitted, separation of duty, access monitoring
Enforcing RBAC	Mostly using DL reasoning	Mostly using rules

Table 1: Approaches to Modeling RBAC in OWL

```
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.
```

3.4.4 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. Some rule examples follow.

The following rules enfore the hierarchy for both the **role** and **activeRole** properties.

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

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

For enforcing *static separation of constraints* we use the following rules

```
{ ?S role ?ROLE1, ?ROLE2.
  ?ROLE1 ssod ?ROLE2.
} => { [] a SSODConflict; subject ?S;
          ssod-role ?ROLE1; ssod-role ?ROLE2}.
```

In order to check if an individual is permitted to perform an action, we check if the permission is associated with any of her active roles

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

3.5 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 a 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. Table 1 provides an overview of the differences between the two approaches.

Both these approaches, however, have a fundamental problem with managing state changes due to the essentially monotonic nature of RDF/OWL [12]. 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.

4. DISCUSSION: BEYOND RBAC

Our larger goal is not just to model RBAC concepts in OWL. This paper is a first step in developing a foundation on which we can build newer ideas for information assurance, including attribute based access control, usage control and

security analysis. This section assesses OWL's suitability for the first and last of these three goals, presenting some of the challenges for modeling them in OWL, and possible approaches to accommodating them.

4.1 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 [35] 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 [30], in general. However, with suitable restrictions (e.g., to a Boolean combination of basic roles), the use of role value maps does not effect decidability or worst-case reasoning complexity [2].

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.

> *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.*

4.2 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 [21]. 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.

type	syntax	description
1	A.r ← D	simple member
2	A.r ← B.r1	simple inclusion
3	A.r ← B.r1.r2	linking inclusion
4	A.r ← B.r1 ∧ C.r2	intersection inclusion

Figure 2: RT has four types of policy rules.

type	OWL encoding
1	D a A-r
2	B-r rdfs:subClassOf A-r
3	*problematic*
4	[owl:intersectionOf (B-r1 C-r2)] rdfs:subClassOf A-r.

Figure 3: RT rules of types 1, 2 and 4 can be easily encoded in OWL. Type 3 rules are problematic.

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 [19] designed to support highly decentralized attribute-based access control.

The RT has four types of statements shown in table 2. 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 3 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 knows 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 reason-

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

Figure 4: Example RT access policy with growth and shrink restricted roles.

```
HQ-marketing owl:equivalentClass [owl:unionOf
    (HR-managers HQ-staff HR-sales
     [owl:intersectionOf (HQ-marketingDelg HR-employee)])].
HQ-ops owl:equivalentClass
    [owl:unionOf (HR-manufacturing HR-managers)].
HQ-marketingDelg owl:equivalentClass HR-manufacturers-access.
HR-employee owl:equivalentClass [owl:unionOf
     (HR-manufacturing HR-managers
      HR-sales HR-researchDev)].
HQ-staff rdfs:subClassOf HQ-marketing;
 owl:equivalentClass [owl:unionOf
  (HR-managers [owl:intersectionOf (HQ-specialPanel HR-researchDev)])].
HR-managers rdfs:subClassOf
    HQ-ops, HR-employee, HQ-marketing, HQ-staff.
HR-researchDev rdfs:subClassOf HR-employee.
HR-manufacturers-access owl:equivalentClass HQ-marketingDelg.
HR-manufacturing rdfs:subClassOf HQ-ops, HR-employee.
[owl:intersectionOf (HQ-specialPanel HR-researchDev)]
  rdfs:subClassOf HQ-staff.
HR-sales rdfs:subClassOf HR-employee, HQ-marketing.
[owl:intersectionOf (HQ-marketingDelg HR-employee)]
   rdfs:subClassOf HQ-marketing.
Alice a HR-managers.
Bob a HR-researchDev.
```

Figure 5: Example RT access policy in OWL with growth and shrink restricted roles.

ers will not guarantee complete reasoning in all cases. We believe, however, that the 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 [21] 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."

How can we model these additional constraints, growth and shrink restriction, using OWL concepts? 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 accessible to those employees with a need to know. A policy in RT is shown in Figure 4. 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 ⊇ HQ.marketing ∧ HR.employee ⊇ 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 ⊇ HQ.ops) TRUE
- **Availability.** Will Alice always have access to the marketing plan? This is true if (Alice ∈ 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 ¬ ∈ HQ.marketing) for a Skolem individual x. FALSE

Can we prove these properties in OWL? 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. As discussed in the next section, alternative techniques are needed to fully evaluate some of these properties. This is not surprising, as the complexity of security analysis in RT has been shown to be EXPTIME-complete [31].

While there are limits to OWL's applicability to security analysis, other techniques show promise. In recent prior work [26], we developed a model checking based approach to analyze the trust management policy language, RT. We've successfully verified several types of security properties, including those shown above. Of course, as noted above, many security properties are in general intractable. However, model checking-based tools can provide a great deal of information contributing to the level of assurance one has in ones policies. By representing the state space in a more compatible form (e.g., OBDDs) and by incorporating appropriate optimization techniques, such as data abstraction and

compositional reasoning, model checking can be employed to effectively handle many cases.

5. RELATED WORK

5.1 Policy languages

Researchers have spent a few decades focusing on policies including discretionary policies, mandatory policies such as Bell and LaPadula policies and more recently trust and privacy policies. Role Based Access Control (RBAC) establishes relations between users–roles and permissions–roles [29]. However it is difficult to apply the RBAC model when roles can not be assigned in advance and it is typically not possible to change access rights of a particular entity without modifying the roles. Using policy languages like Rei and KAoS [34] allow access rights (and the other deontic constructs) to be associated with different credentials and properties of entities, and not roles alone. More sophisticated RBAC models allows delegation between roles [4], by delegating the entire set of permissions associated with a set of roles of the delegator to the delegatee.

Security Policy Language [27], can express several types of complex authorization policies with simple elements implemented by a security event monitor. Delegation Logic [18] has the advantages of tractability, wide practical deployment, and relative algorithmic simplicity, yet has good expressive power. Ponder is an object oriented language for specifying security and management policies [7] where policies are rules defining behavioral choices. It allows definition of positive and negative authorization policies, and information filtering, and a simple delegation model.

Other work on policies include XML-based policies for access control as well as policies for trust negotiation [32]. Furthermore, confidentiality, privacy and trust policies for semantic web are also being investigated in [33]. XACML is a general-purpose authorization policy model and XML-based specification language specified by OASIS [11]. XACML essentially enforced attribute based access control. While it has the benefits of ABAC over RBAC, it suffers from the same limitations that RBAC-based access with XML has - it does not consider the semantics of the policies.

The policy languages described above have some common limitations that make them difficult to extend for open, distributed, dynamic environments. They do not (i) engage sharable semantic domain models; (ii) support reasoning about the obligations or capabilities of other principals; (iii) include the ability to reason over utilities; (iv) have delegation models required by dynamic environments; (v) support justification, advising and negotiation required for greater autonomy.

5.2 RBAC and OWL

There have been some recent efforts to look at OWL as a represenation language for RBAC policies. Di et al [8] 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?"*).

Heilili et al [13] 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.

6. 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.

Acknowledgements

This research was partially supported by grants CNS-0716627, CNS-0716424 and CCF-0524010 from the National Science Foundation, the NSF Cybertrust 05-518 program, and AFRL grant FA8750-07-2-0031.

7. REFERENCES

[1] M. Al-Kahtani and R. Sandhu. A model for attribute-based user-role assignment. *Computer Security Applications Conference, 2002. Proceedings. 18th Annual*, pages 353–362, 2002.

[2] F. Baader. Restricted role-value-maps in a description logic with existential restrictions and terminological cycles. *Proc. DL 2003*.

[3] F. Baader. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.

[4] E. Barka and R. Sandhu. Framework for role-based delegation models. In *Annual Computer Security Applications Conference*, 2000.

[5] S. Bechhofer, F. van Harmelen Jim Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein. Owl web ontology language reference, February 2004. http://www.w3.org/TR/owl-ref.

[6] T. Berners-Lee, D. Connolly, L. Kagal, J. Hendler, and Y. Schraf. 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.

[7] N. Damianou, N. Dulay, E. Lupu, and M. Sloman. The ponder policy specification language. *Lecture Notes in Computer Science*, 1995, 2001.

[8] W. Di, L. Jian, D. Yabo, and Z. Miaoliang. Using semantic web technologies to specify constraints of rbac. *Parallel and Distributed Computing, Applications and Technologies, 2005. PDCAT 2005. Sixth International Conference on*, pages 543–545, 05-08 Dec. 2005.

[9] D. Ferraiolo, R. Sandhu, S. Gavrila, D. Kuhn, and R. Chandramouli. Proposed NIST standard for role-based access control. *ACM Transactions on Information and System Security (TISSEC)*, 4(3):224–274, 2001.

[10] D. F. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramouli. Proposed nist standard for role-based access control. *ACM Transactions on Information and System Security*, 4(3):224–274, August 2001.

[11] S. Godik and T. Moses. OASIS extensible access control markup language (XACML). OASIS Committee Secification cs-xacml-specification-1.0, November 2002.

[12] P. Hayes and B. McBride. RDF Semantics. `http://www.w3.org/TR/rdf-mt/`, 2004.

[13] N. Heilili, Y. Chen, C. Zhao, Z. Luo, and Z. Lin. An owl based approach for rbac with negative authorization. *Lecture Notes in Computer Science*, 4092:164, 2006.

[14] I. Horrocks, P. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean. SWRL: A semantic web rule language combining OWL and RuleML. *W3C Member Submission*, 21, 2004.

[15] S. Jajodia, P. Samarati, and V. Subrahmanian. A Logical Language for Expressing Authorizations. *Proceedings of the 1997 IEEE Symposium on Security and Privacy*, page 31, 1997.

[16] L. Kagal, T. Finin, and A. Joshi. A policy language for pervasive systems. In *Fourth IEEE International Workshop on Policies for Distributed Systems and Networks*, 2003.

[17] O. Lassila, R. Swick, et al. Resource Description Framework (RDF) Model and Syntax Specification. 1999.

[18] N. Li, B. N. Grosof, and J. Feigenbaum. A practically implementable and tractable delegation logic. In *Proc. of IEEE Symp. on Security and Privacy, Oakland, CA, USA, May 2000*, 2000.

[19] N. Li and J. Mitchell. RT: A Role-based Trust-management Framework. *DARPA Information Survivability Conference and Exposition (DISCEX)*, pages 123–139.

[20] N. Li, J. Mitchell, and W. Winsborough. Design of a role-based trust-management framework. *Security and Privacy, 2002. Proceedings. 2002 IEEE Symposium on*, pages 114–130, 2002.

[21] N. Li, J. Mitchell, and W. Winsborough. Beyond proof-of-compliance: security analysis in trust management. *Journal of the ACM (JACM)*, 52(3):474–514, 2005.

[22] D. L. McGuinness and F. van Harmelen. Owl web ontology language overview, February 2004. http://www.w3.org/TR/owl-features/.

[23] T. Moses et al. eXtensible Access Control Markup Language (XACML) Version 2.0. *OASIS Standard*, 200502, 2005.

[24] J. Park and R. Sandhu. The UCON$_{ABC}$ usage control model. *ACM Transactions on Information and System Security*, 5(6), 2007.

[25] A. Pretschner, M. Hilty, and D. Basin. Distributed usage control. *Communications of the ACM*, 49(9):39–44, 2006.

[26] M. Reith, J. Niu, and W. Winsborough. Model checking to security analysis in trust management. In *ICDE Workshop on Security Technologies for Next Generation Collaborative Business Applications*, 2007.

[27] C. N. Ribeiro, A. Zuquete, P. Ferreira, and P. Guedes. SPL: An access control language for security policies with complex constraints. In *Network and Distributed System Security Symposium (NDSS'01)*, 2001.

[28] R. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, February 1996.

[29] R. S. Sandhu. Role-based access control. In M. Zerkowitz, editor, *Advances in Computers*, volume 48. Academic Press, 1998.

[30] M. Schmidt-Schauss. *Subsumption in KL-one is undecidable*. Fachber. Informatik, Univ, 1988.

[31] A. P. Sistla and M. Zhou. Analysis of dynamic policies. *Inf. Comput.*, 206(2-4):185–212, 2008.

[32] A. C. Squicciarini, E. Bertino, E. Ferrari, and I. Ray. Achieving privacy in trust negotiations with an ontology-based approach. *IEEE Transactions on Dependable Sec. Comput.*, 3(1):13–30, 2006.

[33] B. Thuraisingham. Assured information sharing. Technical Report UTDCS-43-06, Computer Science Department, University of Texas Dallas, 2006. to appear as Book Chapter in Security Informatics by Springer, editor: H. Chen.

[34] G. Tonti, J. M. Bradshaw, R. Jeffers, R. Montanar, N. Suri1, and A. Uszok1. 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.

[35] L. Wang, D. Wijesekera, and S. Jajodia. A logic-based framework for attribute based access control. *Proceedings of the 2004 ACM workshop on Formal methods in security engineering*, pages 45–55, 2004.

APENDIX

You can access examples that we have implemented for the two approaches at the following URL

`http://dig.csail.mit.edu/2007/rowlbac/`

For each approach we provide the required OWL files and a readme file describing how the example can be run in CWM, a simple forward chaining reason implemented in Python that uses a special notation for rules expressed in RDF. Instructions for downloading and installing it can be found at

`http://www.w3.org/2000/10/swap/doc/CwmInstall`

- Approach one example http://dig.csail.mit.edu/2007/rowlbac/approach1/
- Approach two example http://dig.csail.mit.edu/2007/rowlbac/approach2/