ABSTRACT
The Semantic Web is an extension of the World Wide Web that has been growing in recent years. One important issue in the Semantic Web environment is access control. Integrating Role-Based Access Control (RBAC) models, which have been accepted as a powerful approach to security management, with the Semantic Web helps to reduce the complexity of Web security management. The Generalized Temporal RBAC (GTRBAC) model combines the key features of the RBAC model with a temporal framework to address situations where processes and functions may have limited time spans or periodic temporal durations, and it is useful for applications with inherent temporal semantics such as workflow-based systems. There have been several attempts to adopt basic components of the RBAC to the Semantic Web using Web Ontology Language (OWL). In this paper, we show how to model temporal constraints and restrictions in GTRBAC using OWL. In order to do this, we define OWL ontologies that represent temporal constraints in GTRBAC and describe implementation of a scalable architecture for specification and enforcement of GTRBAC policies. The applicability of the represented model is shown using a running example.

Categories and Subject Descriptors
K.6.5 [MANAGEMENT OF COMPUTING AND INFORMATION SYSTEMS]: Security and Protection; D.4.6 [SECURITY AND PROTECTION]: Access Controls

General Terms
Security, Design, Languages

Keywords
GTRBAC, OWL, Access Control, Temporal Constraints

1. INTRODUCTION
The Semantic Web is an extension of the World Wide Web that allows knowledge, information and data to be shared on the Web and reused across applications and enterprises. It is important for security frameworks, particularly access control models to capture the heterogeneous and distributed nature of the Semantic Web [12]. A security framework for the Semantic Web needs a language to express security policies [16]. To meet this need, some Semantic Web based policy languages for access control have been developed [16, 17, 18, 19, 20]. Reis is implemented in Prolog with a semantic representation of policies using the Resource Description Framework (RDF) [16, 17]. KAoS uses the DARPA Agent Markup Language (DAML) as the basis for representing and reasoning about policies for web services [18, 19]. Ponder is an object-oriented policy language for the management of distributed systems and networks [20]. These policy languages describe policies over heterogeneous domains and support common understanding among participants who might not use the same information model. However, a policy language without ties to a model gives the designer little guidance. Conversely, a model may not have the organized structure to express all the policy details or may leave important aspects unspecified [2]. Integrating access control models with the Semantic Web helps to reduce the complexity of web security management [12]. To support a model in a policy language there must be some ways to express the components and features of the model in the language. This expression should be natural and leverage thinking along the lines of the model while expressing model details.
Web Ontology Language (OWL) is a standard knowledge representation and specification language for the Semantic Web, making it a natural tool for providing access control in that context [3]. Describing security policies by OWL helps web entities to better understand and share the security policies. The use of OWL to define policies has several 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. Although OWL is not a language for expressing authorization policies, it has successfully been used for this purpose in previous work. Some researchers have used OWL as a representation language for RBAC policies [2, 13, 12, 11, 14, 8]. It has been used for representing RBAC policies, reasoning about RBAC authorizations, handling negative authorizations, and specifying constraints. Given the potential of a language like OWL, it would be good to assess whether OWL is expressive enough to support more complicated access control models. To this end, we chose the Generalized Temporal Role Based Access Control (GTRBAC) model that combines the key features of the RBAC model with a powerful temporal framework [1]. The GTRBAC model allows specification of a comprehensive set of time based access control policies, including periodic as well as duration constraints on role enabling, user-role and role permission assignments, and role activations. It provides an event based mechanism for supporting dynamic access control policies, which are crucial for developing secure workflow-based enterprise applications. In addition, the temporal hierarchies and separation of duty constraints facilitated by GTRBAC allow specifying fine-grained temporal semantics. Because OWL is based on XML, it is a language for which there will be supporting tools.

In this paper, we examine the relationship between the GTRBAC model and OWL and model temporal constraints and restrictions in OWL. To enforce GTRBAC in the Semantic Web context, we define OWL ontologies that represent temporal constraints and restrictions in GTRBAC and show how they can be used to specify and implement temporal constraints. We also propose a scalable architecture for specification and enforcement of GTRBAC policies. Moreover, we show how to enforce restrictions and constraints in GTRBAC using an example scenario.

The remainder of this paper is organized as follows: Section 2 presents a brief description of the OWL and the GTRBAC model. Section 3 discusses the related work. Section 4 describes the example scenario that we use in this paper. In Section 5, we show how to represent temporal constraints in GTRBAC using OWL. In section 6, we model enforcement of constraints using OWL. In Section 7, we describe the system architecture. Finally, Section 8 discusses conclusions related to the study.

2. BACKGROUND
This section reviews the OWL and the GTRBAC model.

2.1 Semantic Web and OWL
In the Semantic Web, ontologies are used to specify a domain of interest that consists of terms representing individuals, classes of individuals, properties, and axioms that assert constraints over them. It provides a structured vocabulary that describes concepts and relationships between them as well as a specification of the meaning of terms used in the vocabulary. Different ontology languages provide different facilities. The W3C standards for ontology languages are based on RDF, which provides a basic capability for specifying graphs with a simple interpretation as a semantic network. 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 or alternatively, a binary predication. The Web Ontology Language OWL [3] is a family of standard knowledge representation language for the Semantic Web based on Description Logic (DL) with a representation in RDF. By using a reasoner we can check whether all of the statements and definitions in the ontology are mutually consistent. However, there is a tradeoff between expressiveness and efficient reasoning. The more expressive the language is, more difficult and less efficient the reasoning is. W3C’s Web Ontology Working Group defines OWL as three different sub-languages: OWL-Lite, OWL-DL and OWL-Full. A defining feature of each sub-language is its expressiveness. OWL-DL may be considered as an extension of OWL-Lite and OWL-Full an extension of OWL-DL.

2.2 GTRBAC
The Generalized Temporal Role Based Access Control (GTRBAC) model is an extension of Temporal RBAC (TRBAC) which is an RBAC extension to address temporal constraints. In particular, GTRBAC model allows one to express periodic and duration constraints on roles, user-role assignments, and role-permission assignments. Numerous activation constraints including cardinality constraints and maximum active duration constraints can lead to further restriction of activation of a role in an interval. The GTRBAC model extends the structure of the TRBAC model such that its features including event and trigger expressions subsume those of TRBAC.

2.2.1 Temporal Constraints in GTRBAC
In GTRBAC, a role can have one of the three states: disabled, enabled, and active. Being in disabled state means that a user cannot acquire the permissions associated with the role. A role in the disabled state can be enabled. When a role is in enabled state, users who are authorized to use the role at the time of the request can activate the role. If a user activates a role in enabled state, the state of the role becomes active. The active state indicates that there is at least one user who has activated the role. If a disabling event occurs, roles in the enabled or active state transit to the disabled state. The model allows the specification of the following types of constraints: temporal constraints on role enabling, user-role, and role-permission assignments, activation constraints, runtime events, constraint enabling expressions, and triggers. Priorities are associated with each event in GTRBAC. In GTRBAC, event expressions, priorities, and status predicates are used to express the constraints.
2.2.2 Periodicity and Duration Constraints on Role Enabling and Assignments

The model uses periodicity constraints to specify the intervals. Periodicity constraints are intervals during which a role can be enabled or disabled, and during which a user-role assignment or a role permission assignment is valid. Generally, periodicity constraint expressions are specified by \((I, P, pr : E)\). The pair \((I, P)\) specifies the intervals during which an event \(E\) takes place. \(E\) can be one of the assignment events: “assign\(_p/\)deassign\(_p\) p to r” or “assign\(_u/\)deassign\(_u\) u to r” or a role enabling event: “enable/disable r”. \(pr\) indicates the priority of event.

The model uses duration constraints to specify durations for which enabling or assignment of a role is valid. In case of an event occurrence, the duration constraint associated with the event validates the event for the specified duration only. Generally, the duration constraint expressions for role enabling and assignment are specified by \([(I, P)|D], D_x, pr : E\), where \(x\) is either \(R, U\), or \(P\), corresponding to events: “enable/disable r,” “assign\(_u/\)deassign\(_u\) r to u,” and “assign\(_p/\)deassign\(_p\) p to r,” respectively. \(D\) and \(D_x\) refer to the durations such that \(D \leq D_x\). The symbol “\(|\)” between \((I, P)\) and \(D\) indicates that either \((I, P)\) or \(D\) is specified. The square bracket in \([(I, P)|D]\) implies that this parameter is optional.

2.2.3 Temporal Constraints on Role Activation

Duration constraints can be applied on role activations whereas periodicity constraints on role activations should not be applied. The duration constraints can be classified into two types: total active duration constraint and maximum duration per activation constraint. The total active duration constraint on a role restricts the number of the role’s activation duration in a given period to a specified value. The total active duration can be specified on per-role and per-user-role basis. Per-role constraint restricts the total active duration for a role, while per-user-role constraint restricts the total active duration for a role by a particular user.

3. RELATED WORK

There have been some efforts to look at OWL as a representation language for RBAC policies. Finin et al. have introduced ROWLBAC, a representation of RBAC in OWL [2, 13]. They propose two different approaches: one maps roles to classes and subclasses, and the other maps roles to values. In the first case roles are represented as class of users, and the role hierarchy relation is mapped to the subsumption relation in OWL. Then it maps SoD constraints to class disjointness constraints in OWL. The second approach is to map classes onto individuals and bind users to classes through the property role. It models constraints through specialized properties e.g. DSoD and SSoD. However, a standard DL reasoner can not detect constraint violations and we need to add rules to the ontology that degrades performance. Knetchel et al. have proposed an approach that uses OWL for reasoning about RBAC authorizations [8]. The model can support both roles and class hierarchies. However, it does not take into consideration SoD constraints. Heilili et al have defined users and roles as classes [12]. In order to handle negative authorizations, they define two corresponding classes for each role, each permission or prohibition has corresponding classes for roles and users. In other words, for each permission, there is a class of roles that has that permission, and then a class of users that has that permission. It is similar for each prohibition. Di et al. have described another approach using OWL to specify the RBAC constraints [11]. Their approach models roles, users, permissions, and session as classes, with properties to relate users to roles and roles to permissions. They also define functional mappings between sessions and roles and specify constraints such as separation of duty constraints, prerequisite constraints and cardinality constraints with OWL. However, in order to specify separation of duty and other constraints rules must be added.

Kolovski et al. have developed a DL-based analysis tool for XACML policies [10]. Their proposed approach represents a mapping between Description Logics and XACML as well as reasoning methods to verify properties of XACML policies. Kagel et al. have proposed a general framework based on semantic web technologies that supports general purpose policy systems and is able to solve mismatches among different policy languages [9]. Knechtel et al. have proposed RBAC-CH, an extension of Hierarchical RBAC [14]. The authors extend Hierarchical RBAC by using a class hierarchy of the accessed objects and present a concept to implement this model in a DL knowledge base using an OWL ontology. The permissions of roles are defined on object classes and then users permissions to objects automatically derived by a reasoning service. Cirio et al. have proposed an access control system for context-aware environments designed using Semantic Web technologies, namely OWL and Description Logic [15]. They adopt the RBAC model and extend it with contextual attributes. The authors have developed a high level OWL ontology to express the elements of an RBAC system and also a domain-specific ontology to capture the features of a sample scenario. They use a DL reasoner to classify users and resources, and verify the consistency of the access control policies. To the best of our knowledge, there is no work on time based access control policies that we address in this paper.

4. EXAMPLE SCENARIO

To illustrate our approach we use a simple scenario from a medical information system that is adopted from [1]. The example is shown in Table 1. In row 1a, the enabling times of DayDoctor and NightDoctor roles are specified as a periodicity constraint. The \((I, P)\) forms for DayTime (9:00 a.m.-9:00 p.m.) and NightTime (9:00 p.m.-9:00 a.m.) are as follows:

\[
\text{DayTime} = ([12/1/2008, \infty], \text{all.Days} + 10. Hours \geq 12.Hours)
\]

\[
\text{NightTime} = ([12/1/2008, \infty], \text{all.Days} + 22. Hours \geq 12.Hours)
\]

In constraint 1b, Adams is assigned to the role of DayDoctor on Mondays, Wednesdays, and Fridays, whereas Bill is assigned to this role on Tuesdays, Thursdays, Saturdays, and Sundays. The assignment in constraint 1c indicates that Carol can assume the DayDoctor role everyday between 10:00 a.m. and 3:00 p.m. Constraint 2b specifies a duration constraint of 2 hours for the enabling time of the NurseInTraining role, but this constraint is valid only for 6 hours after the constraint c1 is enabled. Consequently, once the NurseInTraining role is enabled, Amy can activate the NurseInTraining role at the most for two hours.
Table 1: Example GTRBAC Access Policy for Medical Information System

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a</td>
<td>(DayTime, enable DayDoctor), (NightTime, enable NightDoctor)</td>
</tr>
<tr>
<td></td>
<td>b</td>
<td>(M, W, F, assignAmi, Adams to DayDoctor), (T, Th, S, Su, assignBill, Bill to DayDoctor)</td>
</tr>
<tr>
<td></td>
<td>c</td>
<td>(Everyday between 10am-3pm, assignAmi, Carol to DayDoctor)</td>
</tr>
<tr>
<td>2</td>
<td>a</td>
<td>(assignAmi to NurseInTraining), (assignAmi, Elizabeth to DayNurse)</td>
</tr>
<tr>
<td></td>
<td>b</td>
<td>c1=(6 hours, 2 hours, enable NurseInTraining)</td>
</tr>
<tr>
<td>3</td>
<td>a</td>
<td>(enable DayNurse → enable c1)</td>
</tr>
<tr>
<td></td>
<td>b</td>
<td>(activate DayNurse for Elizabeth → enable NurseInTraining after 10 min)</td>
</tr>
<tr>
<td></td>
<td>c</td>
<td>(enable NightDoctor → enable NightNurse after 10 min, (disable NightDoctor → disable NightNurse after 10 min)</td>
</tr>
</tbody>
</table>

Trigger 3a indicates that the constraint c1 in row 2b is enabled once the DayNurse is enabled. As a result, the NurseInTraining role can be enabled within 6 hours. Trigger 3b indicates that 10 min after Elizabeth activates the DayNurse role, the NurseInTraining role is enabled for a period of 2 hours. As a result, a nurse-in-training can have access to the system only if Elizabeth is present in the system. In other words, once the roles are assumed, Elizabeth acts as a training supervisor for a nurse-in-training. Note that Elizabeth can activate the DayNurse role multiple times within a duration of 6 hours after the DayNurse role is enabled.

5. TEMPORAL CONSTRAINTS IN OWL

In this section we describe OWL ontologies that conceptualize temporal constraints introduced in GTRBAC. The ontology defines restrictions and constraints in GTRBAC including activation constraints, cardinality constraints, and temporal constraints.

We create classes representing basic RBAC components (i.e., user, role, etc) as follows. The users are modeled as an OWL class User. The instances of this class represent the users/subjects on which the policies are defined. The association between a user and a role is defined by the ObjectProperty hasRole (User, Role). Moreover, we can link a user to a permission by the ObjectProperty hasPermission (User, Permission).

User rdfs:subClassOf owl:Thing
hasRole a rdfs:Property, owl:ObjectProperty
rdfs:domain :User;
rdfs:range :Role.
hasPermission a rdfs:Property, owl:ObjectProperty
rdfs:domain :User;
rdfs:range :Permission.

We define three classes Action, Object and Permission to represent actions, objects/resources and permissions respectively. The Permission class binds a user to an action-object using two properties hasAction and hasObject.

Action rdfs:subClassOf owl:Thing
hasAction a rdfs:Property, owl:FunctionalProperty
rdfs:domain :Permission;
rdfs:range :Role.
hasObject a rdfs:Property, owl:FunctionalProperty
rdfs:domain :Permission;
rdfs:range :Object.

In order to represent a role hierarchy (R ≤), we model roles as an OWL class Role, and all the roles in R as instances of this class. The ≤ relation is represent by the OWL property subRoleOf(Role, Role). Furthermore, we define subRoleOf to be transitive by making it an instance of owl:TransitiveProperty class. In this way, a reasoner can infer that if subRoleOf(Ri, Rj) and subRoleOf(Rj, Rk) then subRoleOf(Ri, Rk). For example, we can model the relation between Professor and Assistant Professor by adding to the ontology the property subRoleOf( Assistant Professor, Professor). The association between a role and a permission can be defined by ObjectProperty hasPermission(Role, Permission).

Role rdfs:subClassOf owl:Thing
subRoleOf a rdfs:Property, owl:TransitiveProperty
rdfs:domain :Role;
rdfs:range :Role.
hasPermission a rdfs:Property, owl:ObjectProperty
rdfs:domain :Role;
rdfs:range :Permission.

The rest of this section describes the ontology which conceptualizes temporal constraints defined in the GTRBAC model. We describe the important concepts in this paper.

5.1 Periodic Expressions

Periodic expressions are the basis for representing temporal constraints. Calendar classes represent temporal units (i.e. year, month, week, day, and hour). Using the CompositeCalendar, it is possible to define temporal expression based on different temporal units.

YearCalendar rdfs:subClassOf Calendar
MonthCalendar rdfs:subClassOf Calendar
WeekCalendar rdfs:subClassOf Calendar
DayCalendar rdfs:subClassOf Calendar
HourCalendar rdfs:subClassOf Calendar
CompositeCalendar rdfs:subClassOf Calendar

year a rdfs:Property, owl:ObjectProperty
rdfs:domain : CompositeCalendar;
rdfs:range : YearCalendar.
month a rdfs:Property, owl:ObjectProperty
rdfs:domain : CompositeCalendar;
rdfs:range : MonthCalendar.
week a rdfs:Property, owl:ObjectProperty
rdfs:domain : CompositeCalendar;
rdfs:range : WeekCalendar.
day a rdfs:Property, owl:ObjectProperty
rdfs:domain : CompositeCalendar;
rdfs:range : DayCalendar.
hour a rdfs:Property, owl:ObjectProperty
rdfs:domain : CompositeCalendar;
rdfs:range : HourCalendar.
Interval class is defined with two properties, beginTime and endTime as follows.

Interval rdfs:subClassOf owl:Thing
beginTime a rdfs:Property, owl:FunctionalProperty
rdfs:domain : Interval;
The PeriodicExpression class represents duration, for example 10 hours from 2009-06-09, 10 am.

The PeriodicTime class is conceptualized by hasInterval and hasPeriodicExpression properties.

The PeridociExpression class represents duration, for example 10 hours from 2009-06-09, 10 am.

The PeriodicTime class is conceptualized by hasInterval and hasPeriodicExpression properties.

The PeriodicExpression class conceptualizes a trigger in GTRBAC. It has two subclasses according to the type of restriction; CardinalityRestriction and DurationRestriction class.

It has two subclasses according to the type of restriction; CardinalityRestriction and DurationRestriction class.

The former restricts role activation by using activeCardinality property, while the latter uses activeDuration property.

The CardinalityRestriction is represented by two properties: activeCardinality and defaultCardinality. The MaxNumberOfActivationRestriction and TotalNumberOfActivationRestriction are defined as subclasses of CardinalityRestriction.

The DurationRestriction is represented by activateDuration property. The MaxRoleDurationPerActivationRestriction and TotalActiveRoleDurationRestriction are defined as subclasses of DurationRestriction.

The Constraint classes represent temporal policies in GTRBAC. Each Constraint class is associated with only one event. The ActivationConstraint class is related to activation events and it is described by associatedWith, canDuration, and restriction properties.

An Event class has properties related to role, user, or permission depending on the type of relation being represented.

Similarly, we define other events such as UserRoleDeassignment, RolePermissionAssignment, RolePermissionDeassignment, RoleEnabling, RoleDisabling, RoleActivation, ConstraintEnabling, and ConstraintDisabling.

The Trigger class conceptualizes a trigger in GTRBAC. The class has triggeringEvent and triggeredEvent properties.

5.3 Restrictions and Constraints

The restrictions and constraints, most important concepts in GTRBAC, represent temporal constraints. While there are several types of restrictions, we describe only role activation restrictions. The RoleActivationRestriction class associates with RoleActivation and RoleDeactivation events.

The RoleActivationRestriction class represents a role activation from an event. It has two subclasses according to restriction type (duration or cardinality). ActivationDurationConstraint and CardinalityConstraint which are described by restriction property. The MaxRoleDurationPerActivationConstraint and TotalActiveRoleDurationConstraint are subclasses of ActivationDurationConstraint and the MaxNumberOfConcurrentActivationConstraint and TotalNumberOfActivationConstraint are subclasses of CardinalityConstraint. They are distinguished by owll:allValuesFrom on restriction property.

The TemporalConstraint classes are related to all events except (de)activation events.
The class is divided into two subclasses according to the type of restriction, **DurationConstraint** and **PeriodicityConstraint**. DurationConstraint class has two additional properties, canDuration and eventDuration. PeriodicityConstraint has periodicTime property in order to represent periodic restrictions.

### 6. ENFORCING RESTRICTIONS AND CONSTRAINTS

In this section, we apply the presented model to our example scenario. The followings are policies shown in table 1. The specification in illustrative not exhaustive given space constraints.

**Users**
- Adams a gtrbac:User, Bill a gtrbac:User

**Roles**
- DayDoctor a gtrbac:Role, NightDoctor a gtrbac:Role, NurseInTraining a gtrbac:Role

**Events**
- EnableDayDoctor a gtrbac:RoleEnabling
- EnableNurseInTraining a gtrbac:RoleEnabling
- AdamsToDayDoctor a gtrbac:UserRoleAssignment
- Table1-2b a gtrbac:ConstraintEnabling

**Intervals**
- Intervall a gtrbac:Interval
  - beginTime 20031201
  - endTime 99991231

**Periodic Times**
- DayTime a gtrbac:PeriodicTime
  - interval Intervall
  - hasPeriodicExpression DayTimeExp
- DayTimeExp a gtrbac:PeriodicExpression
  - durationUnit HOUR
  - durationValue 24
  - baseCalendar {AllYears, AllMonths, AllWeeks, AllDays, 10}
- Table1-1b-1-time a gtrbac:PeriodicTime
  - interval Intervall
  - hasPeriodicExpression Table1-1b-1exp
- Table1-1b-1exp a gtrbac:PeriodicExpression

---

It is similar for Table1-1b-2-time and Table1-1c-time.

**Constraints for Row1**
- Table1-1a-(a) a gtrbac:PeriodicityConstraint
  - associatedWith EnableDayDoctor
  - periodicTime DayTime
- Table1-1b-(a) a gtrbac:PeriodicityConstraint
  - associatedWith AssignAdamsToDayDoctor
  - periodicTime Table1-1b-1-time

**Constraints in Row2**
- Table1-2b a gtrbac:DurationConstraint
  - associatedWith EnableNurseInTraining
  - canDuration 6, HOUR
  - eventDuration 2, HOUR

**Triggers in Row3**
- Table1-3b a gtrbac:Trigger
  - triggeringEvent ActivateDayNurse
  - triggeringPostfix for, Elizabeth
  - triggeredEvent EnableNurseInTraining
  - triggeredPostfix after, 10, MINUTE
- Table1-3c-(a) a gtrbac:Trigger
  - triggeringEvent EnableNightDoctor
  - triggeredEvent EnableNightNurse
  - triggeredPostfix after, 10, MINUTE

### 7. SYSTEM ARCHITECTURE AND IMPLEMENTATION

In this section, we present the system architecture and provide an overview of the system components and technologies. We extend our previous implementation architecture of a scalable GTRBAC system to show how to specify policies using OWL DL and reason over the ontologies and how to enforce policies [21]. Our proposed system design is shown in Figure 1. As indicated in the figure, the system includes four main modules: specification module, reasoning module, transformation module, and enforcement module.
The **specification module** facilitates specifying temporal constraints in OWL DL. We use Protégé, a Java tool that provides an extensible architecture for the creation of customized knowledge-based applications [4]. The Protégé-OWL editor enables users to build ontologies for the Semantic Web, in particular in the OWL. In order to reason over the ontologies represented in Protégé-OWL, a DIG [6] compliant reasoner is required. DIG interface provides an implementation-neutral mechanism for accessing Description Logic reasoner functionality and allows a variety of tools such as ontology editors to interact with different DL reasoners in a standard way. DIG 2.0 defines an XML Schema that describes a concept language and accompanying operations to be used over an HTTP based interface to a DL reasoner. We use DIG 2.0 to communicate between the specification module and the reasoning module. The **reasoning module** receives the policies from representation module, validates them, and if there is no error sends them to the transformation module. The validation includes identifying whether policies are consistent or not. In this module, we use RACER(Renamed Abox and Concept Expression Reasoner), a well-known reasoner for the Semantic Web languages [5]. RACER provides highly optimized inference services for sophisticated ontology applications by supporting rules, constraint reasoning, and expressive query answering. Using RACER as reasoner along with Protégé, we check the represented policies to determine if there is any inconsistency between policies.

The **transformation module** parses the validated policies and generates appropriate rules based on those policies. These generated rules are used to evaluate applicable policies and render an authorization decision. In this module, we use Jena, a Java framework for building Semantic Web applications [7]. Jena provides a programming environment for RDF, RDFS and OWL and includes a rule-based inference engine. Finally, the **enforcement module** is the system entity that performs access control by making decision requests and enforcing authorization decisions. As mentioned earlier, policies are represented in OWL, validated, and transformed to rules. These rules are used by GTRBAC engine in the enforcement module to enforce policies.

Our system implementation can be viewed as a traditional RBAC system and a GTRBAC add-on module. An RBAC engine is used to enforce RBAC policy and GTRBAC add-on does not affect the RBAC engine and can be added on any existing RBAC system no matter how it is implemented or enforced. In fact, the GTRBAC add-on module only updates the RBAC policies. Horizontally, the system is divided into three levels: Interface Level, Logical Level, and Database Level. On interface level, the administrators specify/update the policies; and the user can issue access requests to the RBAC engine and get authorization decision from it. At the database level, the RBAC policies are stored as a set of tables in a relational database. At the logical level, the specification module interacts with the reasoning module to check the consistency of the policies and the transformation module translates them to the data structures in the database. The GTRBAC engine and RBAC engine also reside at the logical level. The details of implementation of GTRBAC Engine have been discussed in our previous work [21]. The overall system architecture which is an extension of our previously proposed architecture for GTRBAC implementation, is shown in Figure 2.

As indicated in Figure 2, the system operates as follows.

- The security administrators write policies in OWL files using Protégé-OWL editor.
- The OWL files are sent to RACER using DIG interface.
- RACER performs reasoning and checks the consistency of OWL ontology and loads them into a DL knowledge base.
- Jena framework generates rules based on validated DL knowledge base and loads them into a GTRBAC policy base.
8. CONCLUSION AND FUTURE WORK

Integrating RBAC models with the Semantic Web helps to reduce the complexity of Web security management. There have been several attempts to adopt basic components of the RBAC for the Semantic Web using OWL. The GTRBAC is an extension of RBAC that is useful for applications with temporal semantics. It addresses situations that processes and functions may have limited time spans or periodic temporal durations. In an attempt to represent temporal constraints in the Semantic Web, we have defined OWL ontologies that represent temporal constraints in GTRBAC and show how they can be used to specify and model temporal constraints. Furthermore, we have proposed a scalable architecture for specification and enforcement of GTRBAC policies. An example scenario is used to show the applicability and efficiency of the represented model.

Acknowledgements

This research has been supported by the US National Science Foundation award IIS-0545912. We would like to thank the anonymous reviewers for their helpful comments.

9. REFERENCES


