

A Constraint and Attribute Based Security Framework for Dynamic Role Assignment in Collaborative Environments^{*}

Isabel F. Cruz, Rigel Gjomemo, Benjamin Lin, and Mirko Orsini^{**}

ADVIS Lab – Department of Computer Science – University of Illinois at Chicago
{ifc|rgjomemo|plin|orsinim}@cs.uic.edu

Abstract. We investigate a security framework for collaborative applications that relies on the role-based access control (RBAC) model. In our framework, roles are pre-defined and organized in a hierarchy (partial order). However, we assume that users are not previously identified, therefore the actions that they can perform are dynamically determined based on their own attribute values and on the attribute values associated with the resources. Those values can vary over time (e.g., the user's location or whether the resource is open for visiting) thus enabling or disabling a user's ability to perform an action on a particular resource. In our framework, constraint values form partial orders and determine the association of actions with the resources and of users with roles. We have implemented our framework by exploring the capabilities of semantic web technologies, and in particular of OWL 1.1, to model both our framework and the domain of interest and to perform several types of reasoning. In addition, we have implemented a user interface whose purpose is twofold: (1) to offer a visual explanation of the underlying reasoning by displaying roles and their associations with users (e.g., as the user's locations vary); and (2) to enable monitoring of users that are involved in a collaborative application. Our interface uses the Google Maps API and is particularly suited to collaborative applications where the users' geospatial locations are of interest.

Key words: role-based access control, collaborative applications, dynamic environments, Semantic Web, reasoning

1 Introduction

With the latest trends in collaborative environments, such as Web 2.0 and cooperative projects on grids, more and more resources are being shared by different

^{*} Work partially supported by NSF Awards ITR IIS-0326284, IIS-0513553, and IIS-0812258.

^{**} Primary affiliation: Dipartimento di Ingegneria dell'Informazione, Università di Modena e Reggio Emilia, Italy. Work partially supported by MUR FIRB Network Peer for Business project (<http://www.dbgroup.unimo.it/nep4b>) and Confindustria Modena.

groups and organizations in order to support common tasks. Depending on several factors such as the task, the participants, and data sensitivity, access to these shared resources needs to be controlled and enforced by security policies. The role-based access control (RBAC) model defines roles that have specific privileges on resources and decouples the identity of the users from the resources [15]. In the RBAC model and its variations, constraints can be placed for example on the associations of users with roles or of roles with permissions. When the number of users is high in comparison with the number of roles [1, 2], an automated way to grant permissions is desirable in order to eliminate the burden of manually assigning roles to users. The RBAC model is particularly suited to dynamic task-oriented environments due to its flexibility and policy-neutrality [14], which enables it to express a large range of policies.

In our paper, we investigate a security framework for collaborative applications that relies on the RBAC model. Roles are pre-defined and organized in a hierarchy (partial order). However, we assume that users are not previously identified. Thus, the actions that they can perform are dynamically determined based on their own attribute values and on the values of the attributes associated with the resources. The user’s attribute values can vary over time during a session (e.g., the user’s location), thus enabling or disabling the user’s roles.

We will focus on a scenario associated with the Olympic Games, where not only the venues directly associated with the Olympic Games (e.g., stadiums, gymnasiums) but also tourist attractions in the area (e.g., museums, parks) are resources of interest in our framework. Access to venues and specific places inside the venues depend on the users’ types. For example, some spectators can only take part in the opening ceremony, whereas others can access all swimming events or all track and field events, depending on the tickets they have purchased. In addition to visitors, there are many organizations collaborating with one another and sharing information and services (including police forces, hosting companies, media, and sport organizations) who ultimately serve a large range of visitors as well as the competing athletes and their support teams.

Privileges granted to users depend not only on each particular organization but can also differ among members of the same organization. For example, some members of the escort service for teams and athletes may be restricted to escort out of a specific venue but not out of other venues (a situation similar to taxi drivers in some cities, where a taxi that transports passengers from the city to the airport cannot subsequently pick up passengers at the airport).

Different people will have different privileges depending on their status. For example, members of the Olympic Committee, who have VIP status, will have reserved seating in all competitions, while top officials of the local organizing committee, who also enjoy VIP status, may have non-assigned seating. Police officers will be able to enter any area, but without seating privileges. Children or students under a certain age may be able to join tours of the Olympic Stadium for free, while other people will have to pay a fee. For security reasons access to the Olympic Village is restricted to few people besides the athletes and their

immediate support teams: for example, employees and volunteers specifically assigned to work in that particular area.

In our approach, the roles of each different collaborating organization are structured in a dominance hierarchy where “higher” roles have all the privileges of “lower” roles. The roles associated with all the organizations can be represented as the union of the hierarchies of roles of the single organizations. Some of the roles have fixed and previously known sets of users, such as police, members of the local organizing committee, or the athletes. Other roles have a large number of possible users that cannot be known a priori, for instance journalists, volunteers, and visitors. In this case, constraints on user attribute values can be used to assign the correct role to each user, based on the values of different attributes (e.g., status, credentials, location, organization). Roles are assigned to users depending on the actual values of their attributes (e.g., VIP, journalist, main stadium, NBC). Constraint values in our framework form partial orders and determine the association of actions with the resources and of users with roles. Therefore, users’ actions are dynamically determined based on their own attribute values and on the values of the attributes associated with the resources.

We have designed and implemented a prototype of our access control framework using semantic web technologies. The roles and other entities defined in the RBAC model are represented using the OWL 1.1 language [11], which is a standard language based on Description Logic (DL). Based on previous work, we use two ontologies: the first ontology describes the domain and the second ontology describes the RBAC entities and is partly derived from the first [4]. Reasoning is performed using the Pellet reasoner [5] and is used to implement several functions, such as user to role assignment, separation of duty constraints, symmetry, and class equivalence.

Our model shares some similarities with other approaches including RB-RBAC [1, 2], GEO-RBAC [3, 8], and ROWLBAC [10]. A notable difference is that it has been fully implemented, while the other approaches have not. Therefore, we have leveraged the expressiveness of an actual reasoning mechanism. However, all the other approaches also propose some sort of reasoning. In particular, RB-RBAC uses rules to determine hierarchical roles starting from a partial order of constraints, while GEO-RBAC uses propagation of constraints along the role hierarchy. We extend RB-RBAC by starting from individual partial orders of attribute constraints and then unifying them. In comparison with GEO-RBAC, our framework is more general in that it targets all types of constraints, not only spatial constraints. We also consider resource attribute constraints, whose satisfaction enables or disables the privileges defined on the resources. ROWLBAC, even if not implemented, proposes reasoning as performed by OWL. The most similar approach to our current approach is our former approach, which was also fully implemented using semantic web technologies [4]. However, in that approach, we used a simpler constraint framework and did not explicitly consider spatial constraints.

The paper is organized as follows. In Section 2, we present the security model and in particular, the attribute constraints arranged in partially ordered sets

and their correspondence with the roles. In Section 3, we describe the different types of entailment that our model supports and give examples of some rules of Description Logic that can be used to express security policies. We also show the process by which users are assigned the correct roles by taking into account constraints. In Section 4 we describe the implementation of the access control model including the design choices we have made. Related work is mentioned in Section 5 and conclusions and future work are discussed in Section 6.

2 Security Model

In this section, we describe the different components that make up our framework. We start by extending our scenario and then we describe the different components that are present in our model. Those components, modeled as classes and as constraints, extend the usual RBAC components.

2.1 Scenario

In our scenario, which is a much simplified version of the kind of considerations needed for the Olympic Games, there are four collaborating organizations: *Media*, *Sports*, *HostingCity*, and *Visitors*. The organizations share the same resources and each of them can be modeled separately. The first organization, *Media*, comprises *MediaOperator* and *Journalist*, where *MediaOperator* has privilege *EnterMediaVillage*, to enter a resource that is reserved for media operators and *Journalist* inherits the privileges of *MediaOperator*. *Journalist* has one additional privilege, *EnterPhotoZone*, to enter a special area that is particularly suitable for taking close up pictures of the athletes.

The second organization, *Sports*, comprises *TeamMember* and *Athlete*. The third organization, *HostingCity*, comprises people who take care of all local organizational tasks. The fourth organization, *Visitors*, comprises all the different people who attend the Olympic Games. We model them as an organization so that we can deal with them similarly to the other groups of people. Visitors can have different degrees of importance, spanning from “VIP” (e.g., members of the Olympic Committee) to “normal” (e.g., common spectators). These different degrees of importance correspond to different privileges. Privileges and the overall role hierarchy of our collaboration scenario is shown in Figure 1. The roles that carry more privileges are shown higher in the hierarchy: for example, the *Manager* role contains all the privileges of the roles that are its descendants in addition to its own, while the role *Volunteer*, which is not a descendant of *Manager*, comprises all the roles of *Employee* and of *NormalVisitor* in addition to its own.

We consider that each organization determines how the roles are assigned to their users depending on their attribute values. For instance, in our scenario, visitors have attributes *Importance*, *Age*, and *Location*. The *Visitor* organization assigns the role *SpecialVisitor* depending on the values of these attributes, for

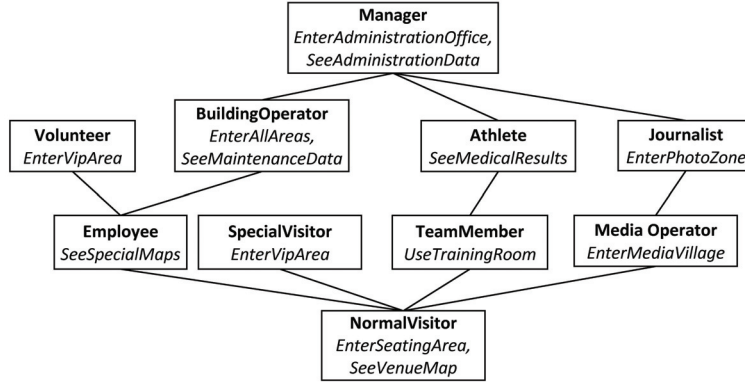


Fig. 1. Roles and privileges for the Olympic Games organizations.

instance if somebody's *Importance* attribute is equal to *VIP*, *Age* is greater than 21, and *Location* is inside *VIPArea*. In our model, *Location* is both an attribute of users, which is used to associate roles with users, and of resources.

The final role hierarchy shown in Figure 1 is derived using simple inference on a description of the organizations and resources using ontologies. The security administrator checks and validates the inference results. Further description of this step will be given in Section 3.

2.2 Framework components

In this section, we explain the conceptual components of our system.

Resource class. This class represents the entities on which different actions are or not allowed (e.g., *SeatingArea*). Resources have associated attributes (e.g., *Capacity* of the Olympic Stadium).

Action class. This class represents the actions that can be performed by users on the resources (e.g., *Enter*).

Privilege class. Objects of this class are pairs $\langle Action, Resource \rangle$. For example, the privilege $\langle Enter, SeatingArea \rangle$ allows some users to enter the seating area.

Privilege attribute constraints. These constraints are pairs $\langle p, a \rangle$, where p is a privilege (e.g., $\langle Enter, SeatingArea \rangle$) and a is a pair $\langle attribute, attributeconstraint \rangle$ (e.g., $\langle isOpen, = true \rangle$) associated with the resource that is part of the privilege (in this case, *SeatingArea*). Attribute constraints are recursively defined as follows:

```

attributeconstraint ::= (attributeconstraint)
                    | RELATIONALOPERATOR constant
                    | NEGATION (attributeconstraint)
                    | attributeconstraint BINARYBOOLEANOPERATOR
                      attributeconstraint
  
```

where a constant can be of different types (e.g., string, number, Boolean, area) and therefore the relational operator (e.g., $=, \leq$) is polymorphic in that it is

able to compare different types (for example, \leq , when used for areas will be equivalent to set containment, \subseteq). Examples of attribute constraints include: $\geq 10 \wedge \leq 18$, and $\neg(\geq 10 \wedge \leq 18)$ and $\leq \textit{SeatedArea}$. The definition of attribute constraint can be further extended.

Role class. This class is a placeholder for all the roles that are defined. Conceptually, a role is a set of privileges. Roles are assigned to users via sessions.

Role attribute constraints. These constraints are pairs $\langle r, a \rangle$, where r is a role (e.g., *SpecialVisitor*), and a is an attribute pair $\langle \textit{attribute}, \textit{attributeconstraint} \rangle$ (e.g., $\langle \textit{Importance}, = \textit{VIP} \rangle$), where *attributeconstraint* is defined as previously. There is a many-to-many relationship between roles and attribute pairs. The role *SpecialVisitor* is assigned to a user if the attribute *Importance* has value $= \textit{VIP}$. When a role attribute constraint refers to spatial attributes, for example, $\langle \textit{Journalist}, \langle \textit{Location}, \leq \textit{MediaVillage} \rangle \rangle$ the role *Journalist* is activated when the user is in the *MediaVillage* (provided that other attribute pairs, if any, are also satisfied).

Session. A user is assigned a session upon entering the system (e.g., *John_680481*). A session is owned by a single user and has a set of roles associated with it. We assume that attribute values associated with resources are not allowed to change during a session. However, attribute values associated with users can change. For example, the location of a user can change during a session, therefore the corresponding attribute *Location* value changes.

2.3 Attribute constraints

As presented in Section 2.2, role attribute constraints denote a many-to-many relationship between roles and attribute pairs. For a role to be assigned to a user, the user’s attribute values must satisfy the attribute constraints. As previously described, the constraints can be expressed in different ways. For instance, a constraint on *Age* can be expressed as a range, for example, ≥ 21 , or a constraint on *Importance* can be expressed as a single value, for example $= \textit{VIP}$. The former constraint would have to be satisfied for someone to have the privilege to enter a bar, whereas the second one would have to be satisfied for someone to access a VIP area.

It is possible to establish a partial order among attribute constraints in the case where an attribute constraint dominates another one. For example, for attribute *age*, ≥ 21 dominates ≥ 18 as someone who is older than 21 is also older than 18. Likewise, for attribute *importance*, $= \textit{VIP}$ should dominate $= \textit{normal}$. In our approach, we interpret the dominance relationship between attribute constraints as a satisfiability relationship. Thus, to say that a constraint a dominates a constraint b , written $b \preceq a$ is tantamount to saying that when a is satisfied, b is also satisfied.

Examples of partial orders are shown in Figure 2. Figure 2.1 shows the constraint for user attribute *Age*. The constraint B_3 is dominated by the constraint B_2 , and the constraint B_2 is dominated by the constraint B_1 ($(\geq 5) \preceq (\geq 18) \preceq (\geq 21)$). Therefore, if the constraint B_1 is satisfied, then the constraints B_2 and B_3 are also satisfied. Figure 2.2 shows the constraint for user attribute

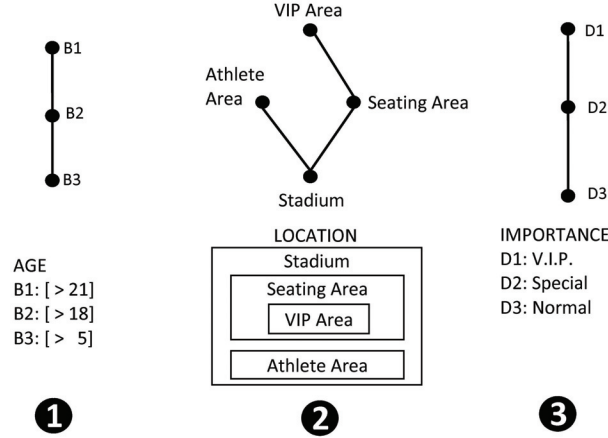


Fig. 2. 1. Age partial order 2. Location partial order 3. Importance partial order.

Location, that is, if the coordinates of a user fall inside one of the regions, then the user is located inside the region. In this case, the dominance relationship represents the spatial containment between regions. If the constraint $\leq VIPArea$ is satisfied, meaning if the user is inside location $VIPArea$, then the constraints $\leq SeatingArea$ and $\leq Stadium$ are also satisfied ($(\leq Stadium) \preceq (\leq SeatingArea) \preceq (\leq VIPArea)$). If the constraint $\leq AthleteArea$ is satisfied, only the constraint $\leq Stadium$ is also satisfied ($(\leq Stadium) \preceq (\leq AthleteArea)$). Figure 2.3 shows the constraint for attribute Importance. The constraint D_3 is dominated by D_2 , which is in turn dominated by D_1 ($D_3 \preceq D_2 \preceq D_1$). Therefore, if the constraint D_1 is satisfied by the user Importance value, then the constraints D_2 and D_3 are also satisfied.

We argue that in a scenario with different collaborating organizations, each having a different role hierarchy, the definition of partial orders of constraints can play an important role. Each organization will have its attributes and respective constraints. However, some of the attributes may be the same, but with different constraints on them. For instance, with respect to Figure 2.1 it is not difficult to imagine different constraints on the Age attribute. If these organizations share their role hierarchies, then they would also share their role attribute constraints. In the next subsection, we discuss the integration of different partially ordered sets of role attribute constraints into one partially ordered set.

2.4 Role-constraints partial order

A role r can be associated with a tuple A of user attribute constraints over distinct attributes. The pair $\langle r, A \rangle$ represents the constraints that must be satisfied to activate the role. Roles are assigned to users, based on the constraints that the user’s attribute values satisfy. In Figure 3 we show three roles and their associate attribute constraints, whose partial orders are shown in Figure 2.

The role *Journalist* is the dominant role represented in the table. Also, each attribute constraint of *Journalist* dominates the corresponding attribute constraint of the other roles, that is, the sets of attribute constraints represented in each row are in *componentwise order* [9]. This type of order can be defined on the tuples of the Cartesian product of partially ordered sets. A tuple of the Cartesian product (e.g., $\langle \geq 21, \leq VIPArea, =VIP \rangle$) dominates another tuple (e.g., $\langle \geq 21, \leq VIPArea, =Special \rangle$) if each element of the first tuple dominates the corresponding element of the second tuple (that is, $\geq 21 \preceq \geq 21$, $\leq VIPArea \preceq \leq VIPArea$, and $=Special \preceq =VIP$).

The cardinality of the Cartesian product of the partially ordered sets of constraints can be much higher than the cardinality of the set of roles. For instance, in the example of Figure 2, there are $3*4*3 = 36$ possible combinations of the different attribute constraints, but likely fewer roles. Therefore, a user may satisfy a set of attribute constraints that does not correspond to any role. For instance, in Figure 3, a user may satisfy the constraints $Age \geq 18$, $Location \leq SeatingArea$, and $Importance = VIP$, which does not correspond to any role. Nonetheless, the user should be assigned the most dominant role possible, that is, *NormalVisitor* [2].

Age	Location	Importance	Role	Privilege
>=21	<= VIP Area	= VIP	Journalist	Enter, reserved best seat
>=21	<= VIP Area	= Special	MediaOperator	Enter, reserved seat
>=18	<= Seating Area	= Normal	NormalVisitor	Enter, seat anywhere

Fig. 3. Attribute constraints and roles.

We will discuss later in the implementation part how this feature has been implemented in our framework.

2.5 Transformation functions

A transformation function can be defined on an attribute to associate the attribute values defined in a certain domain with values on a different domain. For example, given the integer attribute *Age*, the transformation function $child : Age \rightarrow Boolean$ associates values greater than 5 to the Boolean *false* and values up to 5 to the Boolean *true*. Transformation functions are total functions. The domain of a transformation function can be the Cartesian product of several attribute domains, associating a set of attributes values with a single attribute value. As in the GEO-RBAC model [3], an example of a transformation function is a location transformation that associates the geographic coordinates of a user with a logical location (e.g., *OlympicStadiumArea*).

With transformation functions applied to a set of user attributes, the constraints can be defined on the target of the transformation function. Applying

transformation functions to the user attributes can help in simplifying the computation of the constraints and in preserving privacy [7]. Indeed, if a transformation function is applied on an attribute, only the transformed values (logical values) will be computed over the constraints. The real values will be in a certain sense masked. Moreover, through transformation functions, it is possible to map a set of constraints defined on several attributes into simpler constraints, for example into constraints on Boolean values. In our framework we have implemented only the location transformation function.

3 Reasoning

In the last few years there has been a good amount of research in modeling security models for dynamic environments with the use of Description Logic [4, 10, 17]. Toninelli *et al.* [17] use the OWL language and Logic Programming to model the security policies of a pervasive computing environment. Finin *et al.* consider two approaches for modeling the RBAC model with OWL [10]. Cirio *et al.*, whose work we continue, leverage semantic web technologies to help the security administrator define security policies [4].

The expressiveness of OWL allows for a rich representation of rules and relationships between domain entities and for expressing policies. In particular, it is possible to express:

- Equivalence or disjointness between classes of objects. For instance, it is possible to say that two classes are equivalent and therefore they inherit the properties of each other, or disjoint, therefore an object cannot be an instance of both classes. We use the disjointness feature to implement separation of duty constraints. For instance, it is possible to say that an object belonging to class *TaxiDriver* cannot belong to class *Police*.
- Subclass hierarchies, with multiple inheritance. The subclass inherits the properties of the superclass. We use this feature in two ways: 1) to implement the role constraint hierarchy; 2) to create sets of classes in order to specify a common policy for all of them. The classes of a set are placed under a superclass, to which privileges are attached. Through inheritance, the set of the subclasses inherits the privileges attached to the superclass.
- Properties can be of two types: datatype properties and object properties. We use datatype properties to model the constraints and object properties to assign the privileges to the roles. Object properties, in turn, can also be divided into symmetric, anti-symmetric, transitive, anti-transitive, functional and inverse functional properties.
- New classes can be combined from existing classes using intersection, union, and negation.
- Axioms can be written to express policies. For example, to express the fact that some members of the escort service for teams and athletes may be restricted to escort out of a specific venue but not out of other venues.

We use two types of ontologies in our model: the domain ontology and the RBAC ontology.

The *domain ontology* represents the relationships that hold between the entities of the domain. It can be an existing ontology that describes a particular organization. The domain ontology can contain any of the OWL constructs described above. We give an example of a portion of our domain ontology in Figure 4. In the figure, we show the ontology classes *Manager*, *BuildingOperator*, *Employee*, that are in a class/subclass relationship. Some of the relationships between the different classes are represented by object properties, such as *works*. The *RBAC*

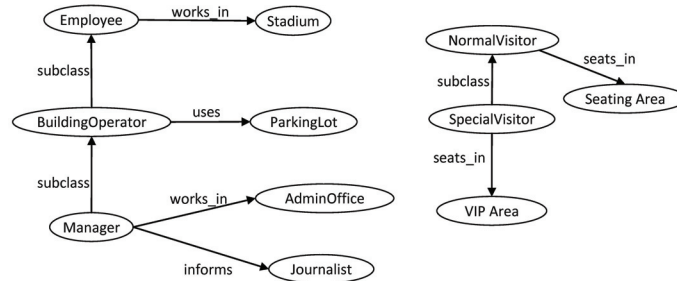


Fig. 4. Domain ontology (portion).

ontology (see Figure 5) has four main classes that represent the main concepts of the RBAC model: *Roles*, *Privileges*, *Actions*, and *Resources* [4]. These main classes are related to one another by object properties. For example the class *Role* has a relationship named *grants* with the class *Privilege*. These properties are useful during reasoning, because they guide the reasoner in classifying each concept of the ontology under the appropriate class of the RBAC ontology.

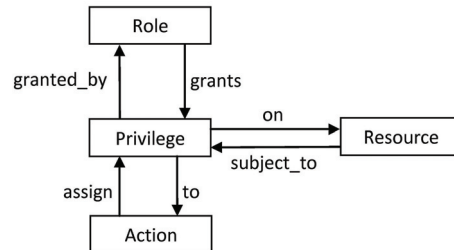


Fig. 5. RBAC ontology.

Two classification tasks are performed: of the user session and of the classes of the domain ontology into classes of the RBAC ontology. A user session is represented as an instance of the class *Thing* and its attribute values are used by the reasoner to classify the user session in the correct role. The classification

of the domain ontology can be performed either by the security administrator or by the the DL reasoner. The latter will classify the different classes of the domain ontology under the classes of the RBAC ontology, following predefined axioms. The axioms are specifications of relationships that must hold between resources [4]. For example, the following rule classifies entities of the domain ontology as subclasses of class *Action* (therefore not in the *Resource*, *Privilege* or *Role* classes):

$$\exists assign.Privilege \sqcap \neg \{Resource, Privilege, Role\}$$

where *assign* is a property in the domain ontology; therefore, given the assertion *assign(Enter, FreeEnter)*, the reasoner classifies *Enter* as a subclass of *Action*.

4 Prototype

We implemented the security model described in Section 2 relying on semantic web technologies. In particular, the access control model and the features of the application domain are modeled using OWL-DL ontologies. The inference capabilities supported by the OWL-DL language enable the association of the ontology with the Pellet reasoner to perform the classification and reasoning tasks described in Section 3. We used Protege 4.0 to write the ontologies, and the Jena API, as an interface to the ontologies. We used the OWL 1.1 language for complex user-defined data types by means of the new *DataRange* constructors. The Pellet reasoner 1.5.2 supports reasoning on the new constructors. In what follows, we describe the classes that we used in the domain ontologies and in the RBAC ontology.

4.1 Domain ontology

In the domain ontology, the entities of the domain are described with OWL classes, data type, and object properties. A figure of a portion of our domain ontology was shown in the previous section.

As mentioned in Section 3, the security administrator defines privileges in the domain ontology. Conceptually, the privileges are pairs of *actions* and *resources*. From a practical point of view, this means augmenting the domain ontology by adding new classes to represent the privileges and actions unless they are already in the domain ontology. The security administrator also creates relationships between the classes of the domain ontology and the new added classes. In Figure 6, we show a portion of this process. The figure has three parts. The RBAC ontology is shown at the top. In the beginning, this is a very simple ontology. The domain ontology is shown on the left and on the right is the ontology that specifies the privileges and actions. The latter can be created by the security administrator or be an existing specification of privileges and actions.

In our example, the security administrator creates two OWL classes: *EmployeeEnter*, to represent a privilege, and *FreeEnter*, to represent the associated

constraint is associated. The range is the XML data type to which the constraint value belongs. We have considered only string and integer data types for now. For instance, to model the constraints on the *Importance* attribute, we first declare a data type property named *Importance*, whose domain is the union of all the roles that have *Importance* as a constraint, e.g., the set $\{SpecialVisitor, NormalVisitor\}$. We declare the range of *Importance* to be the string data type.

(2) Restriction of the values that the attribute can have inside the classes that represent roles or resources. For example, in the class for role *SpecialVisitor*, we restrict property *Importance* to assume only value *special*.

- **RoleConstraint class.** As was mentioned in Section 2, the *RoleConstraint* represents a role and its constraints. We model every *RoleConstraint* as an OWL class. The name of the *RoleConstraint* class is the same as the name of the role, for instance, *SpecialVisitor*. The value of the attribute *Importance* is restricted to assume only the value *special* for the class *SpecialVisitor*. In other words, we are saying that the class *SpecialVisitor* is the class of all objects, whose *Importance* attribute has value *special*. The OWL code for the *SpecialVisitor* class is shown in Figure 7.

```

<owl:Class rdf:about="#SpecialVisitor">
  <owl:equivalentClass>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
        <owl:Restriction>
          <owl:onProperty rdf:resource="#hasAge"/>
          <owl:hasValue>&gt;40</owl:hasValue>
        </owl:Restriction>
        <owl:Restriction>
          <owl:onProperty rdf:resource="#Importance"/>
          <owl:hasValue> special </owl:hasValue>
        </owl:Restriction>
      </owl:intersectionOf>
    </owl:Class>
  </owl:equivalentClass>
</owl:Class>

```

Fig. 7. *SpecialVisitor* role constraint.

- **Session.** At runtime, we add sessions as instances of the OWL class *Thing*, which is the superclass of all the classes of the domain. These instances are augmented with the attributes and values available from the user. The attributes and values of the instance guide the reasoner in the classification. For instance, in Figure 8 we show an instance of the user session with two attributes *Importance*, *Corporation* and values *special*, *HostingCity*, that is classified by the reasoner under the *RoleConstraint* class *Volunteer*.
- **Resource constraints.** With constraints on resource attributes, we have to be able to deal with individual instances, and not with classes of objects anymore. Since each subclass of the class *Resource* can have different instances with different attribute values, we have to identify at instance level the resources that satisfy the constraints. If such resources exist then we can as-

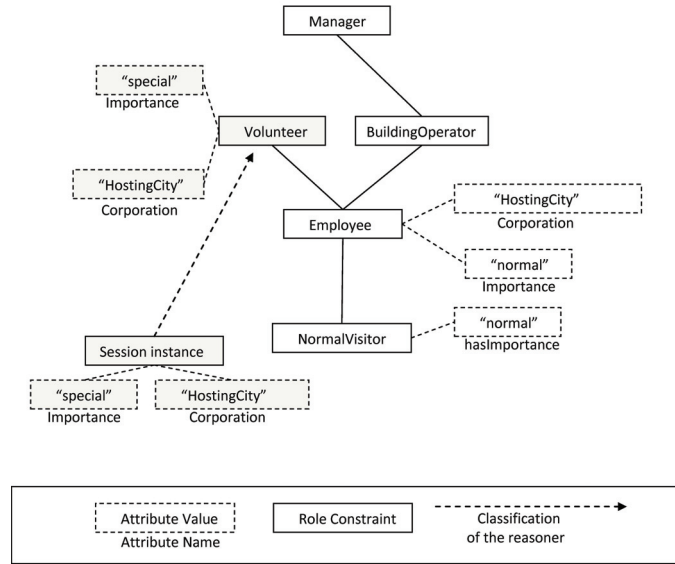


Fig. 8. User session classification.

sociate them with the instance of the user session. This association happens after the instance of the user session has been classified under a role constraint. OWL-DL does not allow for the specification of conditions about actual instances to identify the resources whose attributes satisfy the constraints. Therefore, we use SPARQL queries to verify that such resources exist [4].

4.3 Transformation functions

We have implemented the transformation function for the location attribute using the Google Maps API, which allows to define named areas on the map and symbols to represent people. The symbols can be moved around on the map to simulate the movement of people. If a symbol is inside one of the areas, the API returns the area name, which is used as the *Location* attribute of the user. The transformation functions serve also another purpose in masking from the real attribute values of the user. The access decision is performed on the transformed values and not on the real ones, increasing in this way the privacy of the user. For instance, the location attribute values on which the access decision is made do not show the real coordinates of a user, but a larger area. The location privacy of the user is thus increased [3]. Other transformation functions will be implemented in the future.

4.4 Graphical user interface

For the Olympic Games scenario, we implemented a user interface to illustrate our framework, as shown in Figure 9. It is composed of two parts, the map on

the left and a form to retrieve attribute values when new sessions are created on the right. We have defined eight different areas in the map, associated with different values for the *Location* attribute. The form is used to enter attributes and their values. First, the attribute *Organization* is entered and next a pull down menu allows to choose another attribute for which a value will be entered.



Fig. 9. User interface.

Each session is represented by an icon displaying a person. When a session is created, a unique identifier is appended to the session name. The icon can be dragged and dropped in the map, thus changing the location attribute. The other values of the attributes can also be changed and the session attributes updated. Depending on the values of the attributes, the roles in the session may change. Each time the icon is dropped, the dialog window, which can be seen in the figure, is used to show the enabled and disabled roles associated with that person and the privileges associated with that role.

4.5 Client server architecture

The framework has been implemented in a four-tier client-server architecture:

- **Tier 1: Application web page.** It has been developed with *JavaScript* technology and integrates with the Google Maps API, which also runs JavaScript.
- **Tier 2: Java™ Applet Program.** It is downloaded from the server side and runs on the user's browser. It is responsible for handling the network traffic with the server.
- **Tier 3: Server Side Java™ Program.** It is essentially a network server program, and it is responsible for network server functions, loading the ontology files, and interpreting and processing user requests.
- **Tier 4: Ontologies.** Ontologies are stored in this tier and modeled and maintained independently of the rest of the application.

5 Related Work

Geo-RBAC proposes a model for associating roles with logical location [3, 8]. Logical locations are regions of space defined by real world coordinates and a user can only assume roles that are associated with the location the user is in. In our model, location can be expressed as an attribute of the user along with other attributes, whose values determine the possible roles.

The Proteus system is intended for pervasive computing environments [17]. In Proteus, contexts are defined as intermediaries between entities and operations that they can perform on resources. Contexts are created by data sensed from the environment and reasoning is used to activate permissions on specific resources. Contexts can also inherit constraints from each other. However, Proteus is not role-based.

Kulkarni and Tripathi [13] devise a context-aware access control model. Constraints are defined on different entities of the model, for instance, resources and user attributes. Users can activate personalized permissions in addition to their roles, thus having a somewhat dynamic Role-Permission assignment. Role revocation is also supported, when values of the user attributes no longer satisfy the constraints. Attribute constraints are not arranged in lattices.

ROWLBAC proposes modeling RBAC with OWL [10]. Two different approaches for modeling roles are shown, one where roles are represented as classes and another one where roles are represented as instances. Attribute constraints on role assignments are not modeled, however, and there is no associated system.

RB-RBAC (Rule-Based RBAC) shares some similarity with our approach in that a hierarchy of constraints is mapped to a hierarchy of roles [1, 2]. The rules that associate attributes to roles are arranged in a hierarchy of seniority. When a senior rule is satisfied, the junior rules are automatically satisfied and all the roles produced by the senior rule and the junior ones are assigned to the user. Several other aspects are also considered, including the concept of role hierarchies that are induced by rules. However, they consider just one hierarchy of constraints.

6 Conclusions

The contributions of our paper are summarized as follows:

- We decouple the constraints on the attributes of users from the roles and investigate the relations between hierarchies of attribute constraints and of the roles. Likewise, we decouple the constraints on the resources from their privileges. This simplifies the process of reasoning about users, resources, roles, and privileges.
- We consider dynamic attributes for users, whose values can vary during the same user session. An example includes location, though we offer a unified approach to any attribute type.

- Our model is expressive enough to capture hierarchies both of constraints and of roles and the associated inheritance reasoning as well as reasoning to combine constraints and to infer roles and user sessions.
- We have implemented our framework by exploring the capabilities of semantic web technologies and namely of OWL 1.1 [11] to model our framework and the domain, and to perform reasoning using the Pellet reasoner [5].
- We have adopted a client-server architecture and implemented a user interface whose purpose it twofold: (1) to offer a visual explanation of the underlying reasoning by displaying roles and their associations with users (e.g., as the user’s locations vary); (2) to enable monitoring of the users that are involved in a collaborative application. Our interface, which uses the Google Maps API, is particularly suited to collaborative applications where the users’ geospatial location is of interest.

Future work includes:

- Adding expressiveness to our framework by allowing other types of constraints, namely temporal [12] or more complex constraints. In addition, further exploration of the consequences of componentwise order (or lack thereof) and of the implementation of transformation functions for attributes other than location can be undertaken.
- Investigating reasoning, conflict resolution, and other aspects of merging ontologies of constraints and roles.
- Considering other privacy aspects, in particular when revealing to other organizations the structure of one’s own. Work in privacy-preserving ontology matching [6, 16] needs to be investigated in our particular context.
- Designing a framework for the evaluation of dynamic constraint approaches that will take into account security metrics and the complexity of the evaluation [3] as well as the efficiency of the implementation using semantic web languages and reasoning [10].

References

- [1] M. A. Al-Kahtani and R. Sandhu. A Model for Attribute-Based User-Role Assignment. In *Annual Computer Security Applications Conference (ACSAC)*, pages 353–364. IEEE Computer Society, 2002.
- [2] M. A. Al-Kahtani and R. Sandhu. Induced role hierarchies with attribute-based RBAC. In *ACM Symposium on Access Control Models and Technologies (SACMAT)*, pages 142–148, 2003.
- [3] E. Bertino, B. Catania, M. L. Damiani, and P. Perlasca. GEO-RBAC: A Spatially Aware RBAC. In *ACM Symposium on Access Control Models and Technologies (SACMAT)*, pages 29–37, 2005.
- [4] L. Cirio, I. F. Cruz, and R. Tamassia. A Role and Attribute Based Access Control System Using Semantic Web Technologies. In *International IFIP Workshop on Semantic Web and Web Semantics*, volume 4806 of *Lecture Notes in Computer Science*, pages 1256–1266. Springer, 2007.

- [5] Clark & Parsia, LLC. Pellet. <http://pellet.owldl.com>.
- [6] I. F. Cruz, R. Tamassia, and D. Yao. Privacy-Preserving Schema Matching Using Mutual Information. In *IFIP Conference on Data and Applications Security (DBSec)*, volume 4602 of *Lecture Notes in Computer Science*, pages 93–94. Springer, 2007.
- [7] M. L. Damiani and E. Bertino. Access Control and Privacy in Location-Aware Services for Mobile Organizations. In *International Conference on Mobile Data Management (MDM)*, pages 11–20, 2006.
- [8] M. L. Damiani, E. Bertino, B. Catania, and P. Perlasca. GEO-RBAC: A Spatially Aware RBAC. *ACM Transactions on Information and System Security (TISSEC)*, 10(1):2, 2007.
- [9] M. R. Darnel. *Theory of Lattice-Ordered Groups*. CRC Press, New York, New York, 10016, 1995.
- [10] T. W. Finin, A. Joshi, L. Kagal, J. Niu, R. S. Sandhu, W. H. Winsborough, and B. M. Thuraisingham. ROWLBAC: Representing Role Based Access Control in OWL. In *ACM Symposium on Access Control Models and Technologies (SACMAT)*, pages 73–82, 2008.
- [11] I. Horrocks, P. F. Patel-Schneider, and B. Motik. OWL 1.1 Web Ontology Language Structural Specification and Functional-Style Syntax, 2007.
- [12] J. Joshi, E. Bertino, U. Latif, and A. Ghafoor. A Generalized Temporal Role-Based Access Control Model. *IEEE Transactions on Knowledge and Data Engineering*, 17(1):4–23, 2005.
- [13] D. Kulkarni and A. Tripathi. Context-aware Role-based Access Control in Pervasive Computing Systems. In *ACM Symposium on Access Control Models and Technologies (SACMAT)*, pages 113–122, 2008.
- [14] S. L. Osborn, R. S. Sandhu, and Q. Munawer. Configuring Role-based Access Control to Enforce Mandatory and Discretionary Access Control Policies. *ACM Transactions on Information and System Security (TISSEC)*, 3(2):85–106, 2000.
- [15] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-Based Access Control Models. *Computer*, 29(2):38–47, 1996.
- [16] M. Scannapieco, I. Figotin, E. Bertino, and A. K. Elmagarmid. Privacy Preserving Schema and Data Matching. In *ACM SIGMOD International Conference on Management of Data*, pages 653–664, 2007.
- [17] A. Toninelli, R. Montanari, L. Kagal, and O. Lassila. Proteus: A Semantic Context-Aware Adaptive Policy Model. In *IEEE International Workshop on Policies for Distributed Systems and Networks*, pages 129–140, 2007.