Role-Based Access Control for Model-Driven Web Applications

Mairon Belchior [1], Daniel Schwabe[1], and Fernando Silva Parreiras[2]

[1] Department of Informatics, PUC-Rio.,
Rua Marques de Sao Vicente, 225. Rio de Janeiro, RJ 22453-900, Brazil
[2] Faculty of Business Sciences – FACE, FUMEC University,
Av. Afonso Pena 3880, 30130-009, Belo Horizonte, Brazil
mbelchior,dschwabe@inf.puc-rio.br,fernando.parreiras@fumec.br

Abstract. The Role-based Access Control (RBAC) model provides a safe and efficient way to manage access to information of an organization, while reducing the complexity and cost of security administration in large networked applications. However, Web Engineering frameworks that treat access control models as first-class citizens are still lacking so far. In this paper, we integrate the RBAC model in the design method of Semantic Web applications. More specifically, this work presents an extension of the SHDM method (Semantic Hypermedia Design Method), where these access control models were included and seamlessly integrated with the other models of this method. The proposed model allows the specification of semantic access control policies. SHDM is a model-driven approach to design Web applications for the Semantic Web. This extension was implemented in the Synth environment, which is an application development environment that supports designs using SHDM

Keywords: SHDM, Access Control Model, RBAC, Semantic Web, Ontology.

1 Introduction

Web engineering is a discipline that promotes systematic approaches for dealing with multiple aspects of the process of developing Web applications. Over the years, frameworks like, WebML (Web Modeling Language) [2], UWE (UML-based Web Engineering) [7], OOHDM (Object-Oriented Hypermedia Design Method) [10], Hera [11], and SHDM (Semantic Hypermedia Design Method) [9] have been incrementally improved for dealing with challenges inherent in Web applications.

Access control has long been identified as a necessary feature in applications, notably using models such as Role-based Access Control model (RBAC) [12,3]. Such access control models aim at simplifying security management and at providing constructs for specifying policies.

Although access control models have been investigated over the years, the seamless integration of access control models with Web Engineering frameworks has not gained as much attention as the integration of other models within Web Engineering like domain, navigation and interface models. In this scenario, the question that arises is: What are the connection points of a seamless integration between access control modeling and Web application modeling languages?

This seamless integration enables Web engineers to handle access control elements such as users, resources and rights as first-class citizens. It provides a mechanism for specifying and validating user access to resources in a declarative matter.

The challenge of integrating access control modeling and Web application modeling languages lies in the differences between underlying formalisms of each aspect. Access control models typically rely on formalisms such as the Web ontology language (OWL) as underlying formalism. In contrast, Web application modeling approaches cannot commit to such an expressive language and often rely on RDF to define concepts, properties and relations.

Proposed solutions like [4, 5, 6] use OWL to represent the role-based access control model and, thus, to describe resources. The problem with these approaches arises when a resource in the RBAC model requires the reification of many objects in the domain model. This problem hampers integrated reasoning with both RBAC models and Web application models.

We extend current work by proposing an approach for connecting RBAC models with RDF-based Web application models, such as the Semantic Hypermedia Design Method (SHDM). We treat access control concepts as first-class citizens in the SHDM approach, which enables users to specify, validate and implement access control models seamlessly integrated with domain and business logic models.

We present our approach in this paper as follows. After describing the example we are going to use through the paper in Section 2, we present our approach for integrating access control modeling and Web application modeling in Section 3. We discuss the implementation and proof of concept in Section 4[1]. Section 5 presents the related work and with Section 6 we draw some conclusions and discuss future work.

2 Running Example

To help illustrate the concepts discussed in the paper, we use a running example of a simplified Conference Review Management System, which provides a set of services to run a conference. Activities are carried out by roles, as follows:

a) Conference Chairs (CC) – create a conference, providing name, place, dates, and nominating one or more PC Chairs;
b) PC Chairs (PCC) – responsible for assembling the Program Committee; invite Senior and regular PC Members; assign/de-assign papers for review by PC members; assign/de-assign papers for Senior Reviewers to coordinate reviews from regular reviewers; accept/reject papers;
c) Senior reviewer (SR)– Responsible for overseeing the reviews of assigned papers by regular reviewers; summarize reviews making recommendation to PC Chairs for acceptance/rejection;
d) Reviewer (R) – Responsible for producing a review for each paper assigned to him/her;
e) Author (A) – Responsible for creating/removing submissions; producing final copy of paper if accepted; queries the acceptance status of her/his paper.

[1] A demo of Synth with Access Control, with the running example can be found in `http://www.tecweb.inf.puc-rio.br/navigation/context/o_e1a8b079@0?p=`

A snippet of the Domain Model for this application is shown in Fig. 1**.**

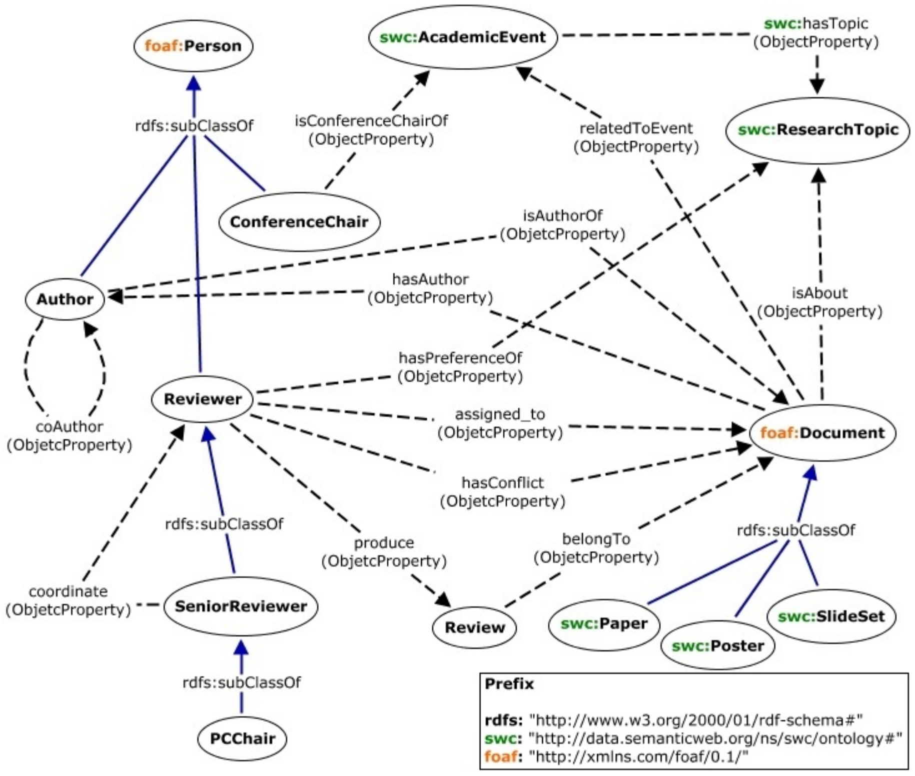
foaf:Person
swc:AcademicEvent
swc:hasTopic (ObjectProperty)
rdfs:subClassOf
isConferenceChairOf (ObjectProperty)
relatedToEvent (ObjectProperty)
swc:ResearchTopic
ConferenceChair
isAuthorOf (ObjetcProperty)
Author
hasAuthor (ObjetcProperty)
isAbout (ObjectProperty)
hasPreferenceOf (ObjetcProperty)
Reviewer
assigned_to (ObjetcProperty)
foaf:Document
coAuthor (ObjetcProperty)
hasConflict (ObjetcProperty)
rdfs:subClassOf
produce (ObjetcProperty)
belongTo (ObjetcProperty)
rdfs:subClassOf
coordinate (ObjetcProperty)
SeniorReviewer
Review
swc:Paper
swc:Poster
swc:SlideSet
rdfs:subClassOf
PCChair
Prefix
rdfs: "http://www.w3.org/2000/01/rdf-schema#"
swc: "http://data.semanticweb.org/ns/swc/ontology#"
foaf: "http://xmlns.com/foaf/0.1/"

Fig. 1. A Domain Model for the Conference Management System

There are many policies applicable in this domain. In the following, we illustrate some that incorporate semantic concepts of the domain, such as "conflict of interest".

1. Reviewers and Senior Reviewers may also be Authors.
2. Nobody is allowed to see any information about a paper for which there is a conflict of interest.
3. A conflict of interest for a reviewer R with respect to a given paper occurs if
   - R is an author of the paper;
   - R is in the same Department/Lab/Group/Project as co-author S of the paper;
   - R and S have co-authored another paper submitted to the same conference.
4. A Reviewer can only see other reviews of a paper assigned to her/him if she/he has already entered his/her own review for that paper.

Next we present our proposed RBAC model, and then discuss how it has been integrated with the SHDM design method.

3 Integrating SHDM and Role-Based Access Control

An access control mechanism is usually composed of two parts: *authentication*, responsible for verifying user identity, and *authorization*, responsible for granting user access to system objects.

Existing approaches for modeling access control share a core set of concepts: they essentially refer to a *User* in the role of *Subject*, having some kind of *Permission* to execute an *Operation* on an *Object*. A well-known approach for modeling access control rules is the Role-based Access Control (RBAC) [12] model, which allows defining policies based on the subject like competence, interest and privilege.

With the rise of the Semantic Web, several techniques like [4, 6, 5] have investigated the usage of OWL for representing RBAC models, which has led to multiple alternatives for creating a formal representation of RBAC models. In this section, we describe the modeling approach we propose for integration with a Web Engineering method. The reader might notice that the solution proposed for integrating RBAC models and a Web Engineering framework is independent of the approach used for the latter. Therefore, it is possible to apply the proposed technique in other configurations of RBAC and other Web Engineering frameworks.

3.1 The RBAC Model

An approach for modeling RBAC using OWL is proposed by [5], named RowlBAC. They develop two different approaches to represent the RBAC model using the OWL language, including the concepts of subject, role, object, actions and associations defined by roles. The first approach specifies roles as classes and subclasses, while the second approach, the roles are represented as properties. The model of access control implemented in this work is based on the second approach, as it is simpler and more concise.

Fig. 2 depicts a diagram with the main concepts and properties of the second approach. Roles are represented as instances of the class *Role*. A role hierarchy is defined by the property *subRole(Role, Role)*, which is an owl:TransitiveProperty. The property *supRole(Role, Role)* is the inverse of (owl:inverseOf) *subRoleOf(Role, Role)*. The relation between a *Subject* and a *Role* is defined by the property *role(Subject, Role)*. The binding between access permissions and roles is done by the properties rbac:permitted and rbac:prohibited, that connect a role (rbac:Role) to a permitted action (rbac:PermittedAction), and to a prohibited action (rbac:ProhibitedAction), respectively.

Not all components of the RBAC model can be specified using OWL DL. Rules in N3Logic were added to the RBAC ontology to define the hierarchy of roles, the static and dynamic constraints, activation and deactivation of roles and permissions associated with roles.

In this paper, we extend the SHDM method to include a new model, the Access Control Model as an addition to the existing Behavior Model.

The Access Control Model is composed of primitives responsible for the description of concepts related to access control, such as subject, the subject's role, permission, object and operation.

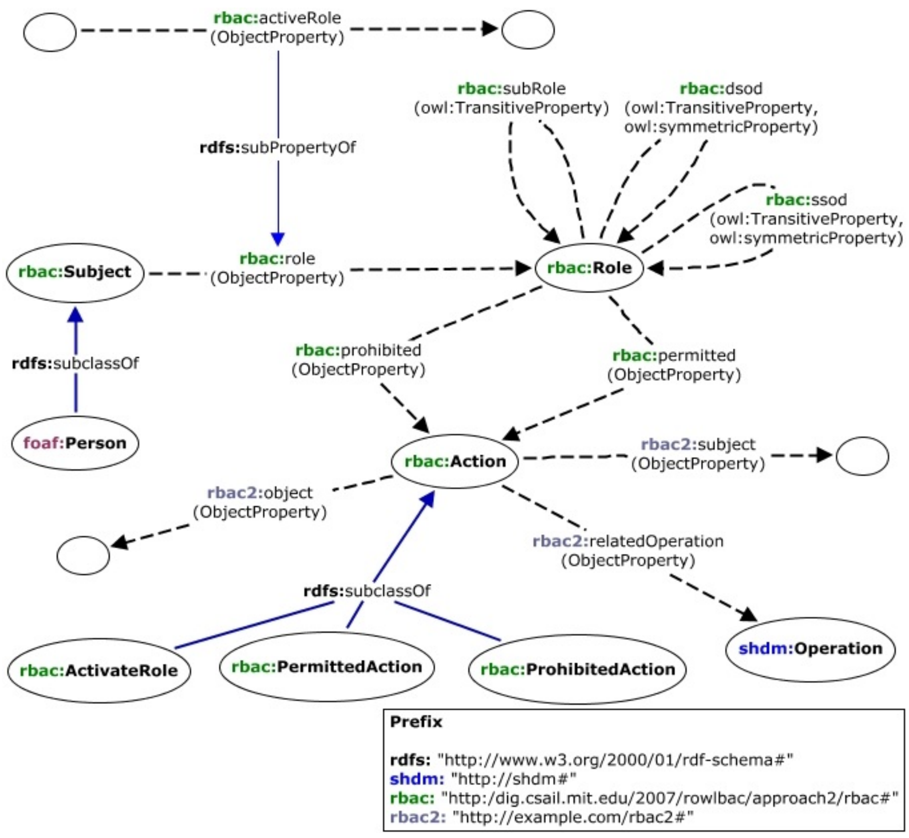
rbac:activeRole (ObjectProperty)
rdfs:subPropertyOf
rbac:subRole (owl:TransitiveProperty)
rbac:dsod (owl:TransitiveProperty, owl:symmetricProperty)
rbac:ssod (owl:TransitiveProperty, owl:symmetricProperty)
rbac:Subject
rbac:role (ObjectProperty)
rbac:Role
rdfs:subclassOf
foaf:Person
rbac:prohibited (ObjectProperty)
rbac:permitted (ObjectProperty)
rbac:Action
rbac2:subject (ObjectProperty)
rbac2:object (ObjectProperty)
rbac2:relatedOperation (ObjectProperty)
rdfs:subclassOf
rbac:ActivateRole
rbac:PermittedAction
rbac:ProhibitedAction
shdm:Operation
Prefix
rdfs: "http://www.w3.org/2000/01/rdf-schema#"
shdm: "http://shdm#"
rbac: "http:/dig.csail.mit.edu/2007/rowlbac/approach2/rbac#"
rbac2: "http://example.com/rbac2#"

Fig. 2. The RBAC vocabulary

We have modified the ROWLBack model as follows:

- The property rbac:subject is no longer an owl:FunctionalProperty, because a certain action (rbac:Action) can be performed by more than one user (rbac:Subject) who has permission to perform this action. For example, the actions rbac:createReview, rbac:editReview, rbac:downloadPaper and rbac:context can be performed by myconference:DanielSchwabe. This property became rbac2:subject in our model;
- The property rbac:object also ceased to be an owl:FunctionalProperty because an action (rbac:Action) may be related to more than one object (rbac:Object). For example, the action rbac:createReview can be applied to rbac:paperA, rbac:paperD and rbac:paperM as objects of this action. This property became rbac2:object;
- The property rbac2:relatedOperation was added to the model to represent which operation (shdm:Operation) defined in the Operation Model of the SHDM is being controlled;
- Classes rbac:PermittedRoleActivation and rbac:ProhibitedRole Activation are not used by this model, therefore they have been removed;

The property rbac2:relatedOperation is used to represent which operation (shdm:Operation) defined in the SDHM Operation Model is being tracked. The code below shows an example of the definition of this property using N3.

```
rbac:createReview rbac2:relatedOperation shdm:createReview .
```

The property shdm:relatedAction was added to the SHDM Operation Model to represent that an shdm:Operation resource has a corresponding rbac:Action, which means that the shdm:Operation is controlled by the application using the primitives of the RBAC access control model.

The association between Roles in the RBAC model and the classes in the Domain Model of SHDM is achieved using the rdfs:subClassOf property, as illustrated in **Fig. 2** with the foaf:Person class.

3.2 Defining Rules

In order to identify the permission to execute a certain action (rbac:Action), we have used N3Logic rules following the same approach as Finin et al. N3Logic is a formalism that allows rules to be incorporated in RDF (Resource Description Framework) [13]. N3Logic uses the syntax Notation3 (or N3), and extends the RDF model to include a set of predicates, e.g., implications, quantified variables, nested graphs, functions and built-ins.

Our approach adds the rule shown below to check if an action (rbac:Action) has an object (rbac:Object). If so, this action is of type rbac:ActionWithObject. The N3Logic rules below, on the right check whether a subject has the permission to perform a certain action on a certain object.

```
   #Check if action has object
   {       ?A a ?RACTION ;
                      rbac2:subject ?S ;
                      rbac2:object ?O .

           ?RACTION a rbac:Action .
           ?S a rbac:Subject .
           ?O a rbac:Object .

} => { ?A a rbac:ActionWithObject } .
```

```
#Permission checking
{?A a ?RACTION ;
                rbac2:subject ?S ;
                rbac2:object ?O .
        ?RACTION a rbac:Action .
        ?S a rbac:Subject .
        ?O a rbac:Object .
        ?Role rbac:permitted ?RACTION .
        ?S rbac:activeRole ?ROLE .
        ?RACTION rbac2:object ?O
        ?A a rbac:ActionWithObject .

} => { ?A a rbac:PermittedAction;
        rbac2:subject ?S;
        rbac2:object ?O
        rbac2:action ?RACTION } .
```

3.3 Modeling Rules for Policies

An access policy is a set of rules that are evaluated to determine whether a user has the right to access a given object. The access policy specifies who is allowed to perform what action on which object depending on (i) properties of the user who made the request, (ii) object properties, (iii) parameters of the action, and (iv) background factors (such as time) [14].

The class rule:Rule reifies the concept of access control policies. It has properties of type DatatypeProperty such as rule: rule_name: defines the name of the rule; rule: rule_title: define a title for the rule; rule: rule_code: defines the specification of the rule; rule: rule_language: defines the language used to specify rules.

Besides these, we consider three additional properties: properties:rule_role, rule:rule_action and rule_object that are used to identify, respectively, the role (rbac:Role), action (rbac:Action) and the object of the action (rbac:Object) on which the rule applies. Examples of rules specifying access policies are shown in section 4.5.

4 Implementation Architecture

A modular software architecture for Access Control was designed as shown in **Fig. 3**. The architecture is divided into two parts: the Authentication Mechanism that is responsible for identifying the user to the system, and the Authorization Mechanism that determines what the user is allowed to do within the system. There are three modules: Authentication Module, Permission Module and Inference Module, represented as gray boxes; white boxes represent the components of the architecture.

The Authentication Module is responsible for performing an authentication protocol, such as FOAF + SSL[2] protocol or OpenID protocol[3]. The Permission

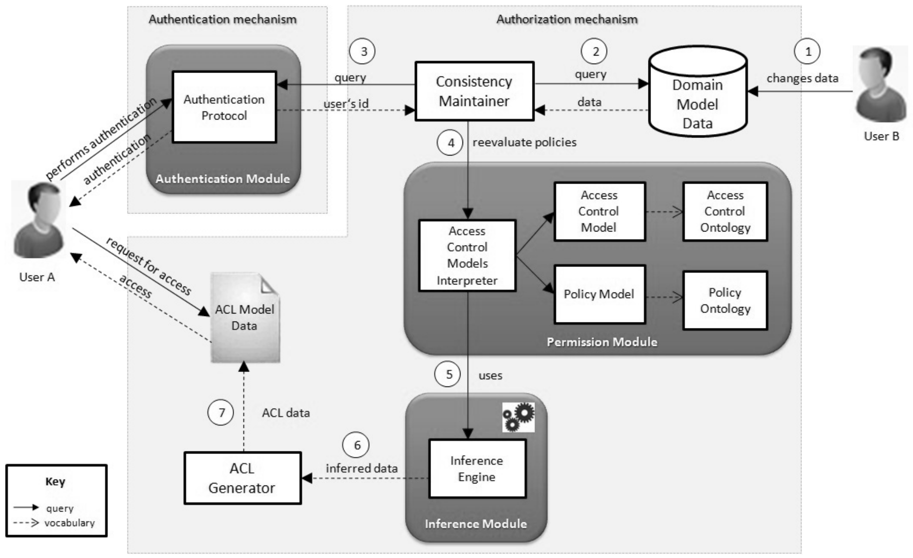

Fig. 3. Conceptual Architecture for Access Control

[2] See http:// www.w3.org/wiki/Foaf+ssl
[3] See http://openid.net

Module should be able to define all access control concepts and policies of the application. The Access Control Model Interpreter maintains and interprets the Access Control and Policy models described by their corresponding ontologies.

The inference module includes a rule inference engine responsible for inferring new facts from existing facts, which effectively evaluate the access control rules. We used the Euler proof engine[4] that supports N3 rules as the inference engine.

The proposed access control system implemented in this work computes the permissions specified in the policies to generate an Access Control List (ACL) [8], represented in the ACL model. This allows for efficient runtime execution, since permissions are read directly from the ACL to decide about authorization, which was previously generated running the reasoner only once.

To be authenticated, a user must provide her credentials to the Authentication Protocol. If the information given is correct, the user is authenticated and the URI representing this user is stored in the application session.

After authentication, the user may request permission to access some resource controlled by the application by providing what operation the user wants to perform on which object, if it exists. The application will check if a permission is present in the ACL model, and if so, the user is authorized to perform the operation. Otherwise, an error message will appear to the user. The ACL needs to be updated whenever a user changes any data. When this occurs, the Consistency Maintainer component of the architecture is triggered, to maintain the permission's consistency in the ACL Model, re-evaluating the policies applicable for that user, and the ACL is regenerated by the ACL Generator component, materializing new inferred permissions. The steps for policy reevaluation are shown in Fig. 3.

The OpenID protocol was used for Authentication. It was chosen because it has support for popular services such as MyOpenID, Google, Yahoo!, etc.

4.1 Integration in the Synth Development Environment

Synth is a development environment for building applications that are modeled according to SHDM. It provides a set of modules that receives, as input, models generated in each step of SHDM and produces, as output, the hypermedia application described by these models. Synth also provides an authoring environment that facilitates the adding and editing of these models through a GUI that can run on any Web browser [1]. Synth was implemented with Ruby on Rails, which is an MVC framework for Web applications development. With this work, the Synth architecture was extended to include the Access Control Module.

4.2 Software Architecture of Synth

The software architecture of Synth consists of five modular components: domain, navigation, behavior, interfaces, and persistence modules. They are responsible for

[4] See `http://eulergui.svn.sourceforge.net/viewvc/eulergui/trunk/eulergui/ html/documentation.html`

maintaining and interpreting each of the models generated in each phase of SHDM method. Each module is composed by a model described in a corresponding ontology in RDFS or OWL, and an interpreter that gives semantics to the models, in addition to the basic semantics of RDFS and OWL, in which they are represented [1]. These modules work together, interpreting their models and communicating with each other, in order to generate the application runtime in accordance with the definitions of each model.

The Access Control module was embodied in the Synth architecture and is responsible for generating the authorization decisions for the application. It maintains and interprets the Access Control and Policy models described by their corresponding ontologies. The Behavior Model Interpreter was extended to handle the Access Control List, checking the permissions in ACL Model whenever an operation is performed. The existing Behavior Model in SHDM, as implemented in Synth, already allows defining a pre-condition to activate an Operation; applying the ACL permissions was implemented simply as a pre-loaded pre-condition on all operations.

Fig. 4 shows a conceptual view of the extended Synth software architecture. The gray boxes represent the modules and the white boxes represent the components of each module. The light gray box is the Access Control Module included in the architecture.

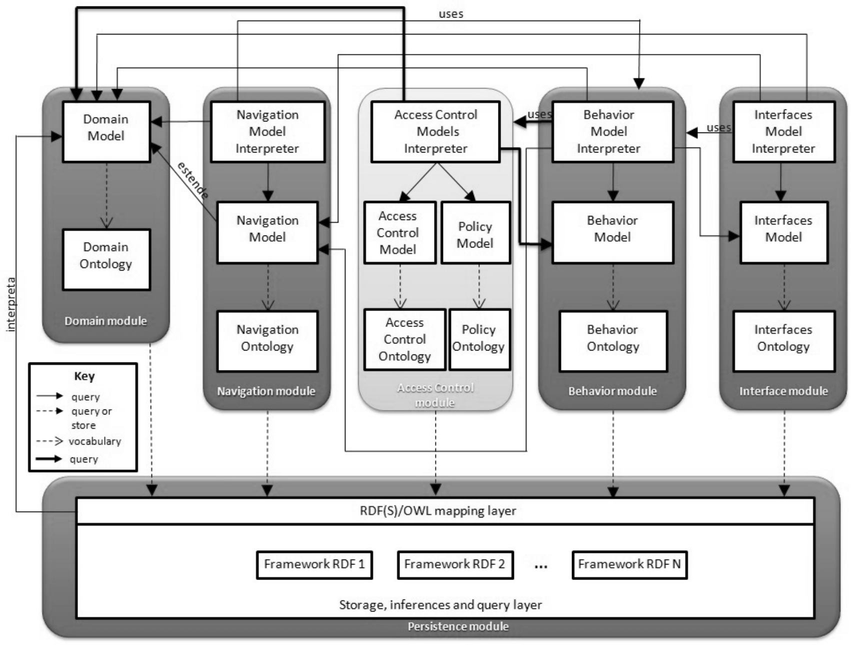

Fig. 4. Conceptual view of the Synth software architecture extended with the Access Control Module

The Behavior Module handles all business logic operations of the application. The Behavior Model Interpreter uses the Access Control Module to determine whether or not an operation can be executed, and the Access Control Model Interpreter queries the

Behavior Model to obtain a set of RDF resources related to the activated operation and its parameters. Similarly, the Access Control Model Interpreter queries the Domain Model to obtain application domain data on which the access control policies are applied.

4.3 Permissions Generation

The generation of access control list (ACL) is achieved in the following steps:

1. A set of possible authorization questions testing if a user has permission to perform an operation on a particular object is generated, by activating all possible roles assigned to this user;
2. Rules in N3Logic to create the ACL resources are generated;
3. The Euler proof engine is activated taking as input all application data and all rules in N3Logic defined in authoring environment.
4. The permissions returned by inference engine containing the RDF resources in the ACL model are added to Synth's database.

The ACL rules in step 2 are generated as follows:

For each access control rule defined in authoring environment with a consequent stating the action is an rbac:PermittedAction, an ACL rule is created with the code whose structure follows the example shown below. The statement "<Access_Control/rulesResult.n3> log:semantics F" associates the inferred triples to F, according to the N3Logic semantics [13].

The sentences (a) "A rbac:action RACTION." and (b) "A a rbac:PermittedActionRBAC." are added to assert that (a) the action is used by the ACL Model and that (b) this action must be permitted by the RBAC Model.

```
@forAll F, S, O, A, RACTION .
{ <Access_Control/rulesResult.n3> log:semantics F .
        F log:includes {
                A a rbac:Action ;
                        rbac2:subject S ;
                        rbac2:object O.

                A a rbac:createReview .
                S rbac:activeRole rbac:reviewer_role .
                O a foaf:Document .

                A a rbac:PermittedActionRBAC .
                A rbac:action RACTION .

                A a rbac:PermittedAction .
        } .
} => { [ a acl:Authorization ;
                acl:mode RACTION ;
                acl:agent S ;
                acl:accessTo O ] } .
```

Notice that if F doesn't include the statement "A a rbac:PermittedAction.", no ACL triples are generated. When accessing the ACL, the absence of authorization is taken as failure.

Occasionally an action can be inferred as permitted and prohibited at the same time. For example, suppose a person is assigned to both reviewer and author roles. Suppose also that there is a policy that defines that an author cannot view the navigational context that lists the papers a reviewer can review, and there is another policy that defines that reviewers can access all navigational contexts of the application. Therefore, when this reviewer is authenticated, all the roles assigned to him will be activated, and then when he tries to navigate to the context described above, whose navigation's semantic is given by the operation shdm:context, such action will be both an rbac:PermittedAction and an rbac:ProhibitedAction at the same time. There are two approaches – conservative and liberal - to deal with this situation to decide which permission should be given. The liberal approach chooses the prohibited action (i.e., everything is permitted unless explicitly stated otherwise) while the conservative approach (everything is denied unless explicitly stated otherwise) selects the permitted action. A parameter in the ACL generation module allows choosing one of these approaches.

4.4 Policy Examples

The policies stated in section 2 can be now defined using N3Logic rules, as used in the Synth environment.

The policy that a Reviewer is not allowed to review a paper by an author from the same institution (one kind of conflict of interest) is defined as follows (left column).

```
{   ?A a rbac:Action ;
          rbac2:subject ?S ;
          rbac2:object ?O .
    ?A a rbac:createReview .
    ?S rbac:activeRole
    rbac:reviewer_role .
    ?O a foaf:Document .

    ?S myconference:assigned_to ?O .

    ?AUTHOR rbac:role rbac:author_role
    .
    ?AUTHOR myconference:isAuthorOf
    ?O .

    ?AUTHOR myconference:memberOf
    ?I1 .
    ?S myconference:memberOf ?I2 .

    ?I1 log:uri ?URI1 .
    ?I2 log:uri ?URI2 .
    ?URI1 log:equalTo ?URI2 .

} => { ?A a rbac:ProhibitedAction } .
```

```
{ ?A a rbac:Action ;
          rbac2:subject ?S ;
          rbac2:object ?O .
  ?A a rbac:createReview .
  ?S rbac:activeRole rbac:reviewer_role
  .
  ?O a foaf:Document .

  ?S myconference:assigned_to ?O .
  ?AUTHOR rbac:role rbac:author_role .
  ?AUTHOR myconference:isAuthorOf
  ?O .
  ?AUTHOR myconference:isAuthorOf
  ?O2 .

  ?O log:uri ?URI1 .
  ?O2 log:uri ?URI2 .
  ?URI1 log:notequalTo ?URI2 .
  ?S myconference:isAuthorOf ?O2 .

} => { ?A a rbac:ProhibitedAction } .
```

The right column shows another similar policy regarding conflict of interest, which states that a reviewer cannot review a paper of a co-author of his/hers, would be very similar, looking at the myconference:isAuthorOf property for two different papers, O and O2.

A third example is the policy whereby a Reviewer can only see other reviews of a paper assigned to her/him if s/he has already entered his/her own review for that paper. This is stated as

```
  {       ?A a rbac:Action ;
                    rbac2:subject ?S ;
                    rbac2:object ?0 .
          ?A a rbac:context .
          ?S rbac:activeRole
rbac:reviewer_role .
          ?O a myconference:Review .

          ?REV a myconference:Review .
          ?S myconference:produce ?REV .
          ?REV myconference:belongTo ?D .
?D a foaf:Document .
?O myconference:belongTo ?D .
?S myconference:assigned_to ?D
.

} => { ?A a rbac:PermittedAction } .
```

4.5 Evaluation

We carried out some preliminary evaluations to determine the overhead of Access Control in Synth. The performance tests were run on a Intel Core i5 CPU M 450 2.40 GHz with 4GB of RAM using Windows 7 Professional 64-bit. Fig. 5 depicts the performance time to query the ACL Model 100 times to access the permission for random access requests for 10 different contexts, where the ACL Model had 217 resources. The fluctuations in the beginning can be attributed to “start up” effects of the simulation as it was run in several separate runs.

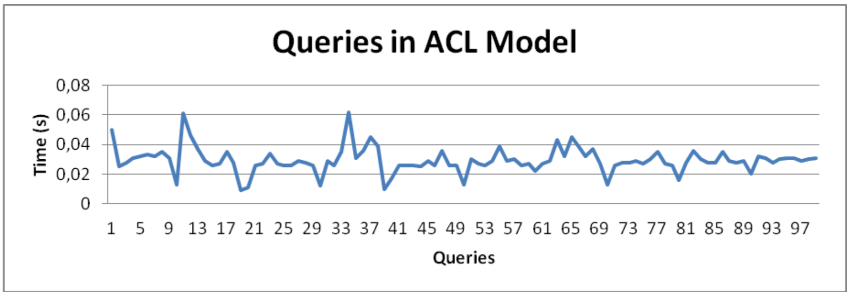

Fig. 5. Time to query the ACL Model

Fig. 6 shows the time to execute all policies (13 plus those used by the RBAC model and those used for ACL generation) access control rules in the example application 50 times by adding a constant number of resources in each run. As expected, the evaluation time grows with the number of data elements added to the database. Note that we currently use a naïve policy re-evaluation strategy.

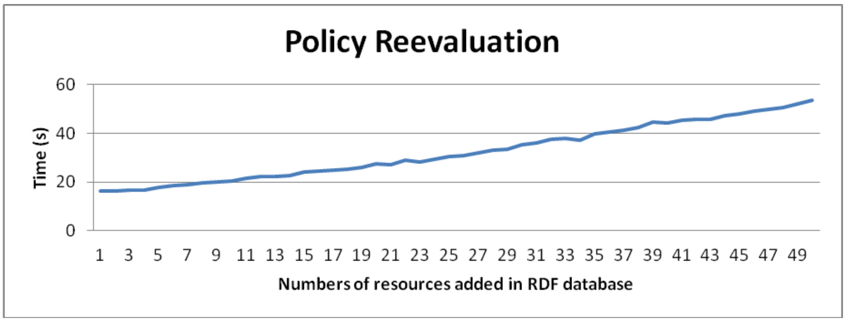

Fig. 6. Time to reevaluate all policies after a change in the database

5 Related Work

Several researchers have investigated the integration of access control with languages related to Web Engineering. In this section, we compare them with our proposal.

Mühleisen et al [15] developed an access control system that allows the use of rules defined in a language created by the authors based on SWRL (Semantic Web Rule Language). However, this language does not follow the RBAC principles and does not allow for modeling any further aspects of the system.

Hollenbach et al [16] developed a system to control access based on ACLs. Although it is possible to model domain aspects, this system does not support the formulation of access control policies.

Finin et all [5] presented two approaches to express the components of the RBAC model using OWL. The main disadvantage of this approach is that each change on the authorization requires a new execution of the inference rules, making the authorization process costly.

Ferrini et al [4] also modeled the RBAC model using OWL and access control policies using XACML in an integrated manner. However, both approaches commit to the principles of OWL, which restricts applications when the closed-world assumption is required.

Knechtel et al [6] show an approach to model an extension of the RBAC model called RBAC-CH, using OWL. RBAC-CH is an access control model that extends the RBAC model by adding a hierarchy of object classes. To evaluate policies using this approach, the inference engine based on OWL is executed once, and the inferred axioms are the available at runtime. However, the authors do not take into account the static and dynamic constraints of the RBAC model, nor do they allow rules for building policies.

Finally, it should be noted that none of the approaches above has been integrated with a Web Engineering method, and is supported by an authoring environment.

6 Conclusions

In this paper, we have presented a novel approach for handling the integration of role-based access control and Web Engineering frameworks. By providing the bridges between these two approaches, we enable Web engineers to specify access control policies at the same level of domain knowledge as well as navigation knowledge. Additionally, we allow for access control models and other Web Engineering models to evolve independently. Our contribution extends the body of knowledge in the field by identifying the links between RBAC and Web Engineering and providing mechanisms for representing the underlying constraints.

There are several extensions we continue to work on. First, extending the policy model to be able to handle dynamic policies, which cannot be evaluated at runtime; second, we want to investigate more opportunistic strategies for incremental policy re-evaluation, so that only affected policies are re-executed when data changes. A third extension can also optimize the ACL, by including the actual role in the ACL itself. Finally, we want to provide a better authoring environment for policies to help end users understand complex sets of policies.

Acknowledgments. Daniel Schwabe was partially supported by CNPq (WebScience INCT). Mairon Belchior was partially supported by a grant from CAPES. Fernando Silva Parreiras was partially funded by FP7-PEOPLE-2009-IRSES, under grant number 24761.

References

1. de Souza Bomfim, M.H., Schwabe, D.: Design and Implementation of Linked Data Applications Using SHDM and Synth. In: Auer, S., Díaz, O., Papadopoulos, G.A. (eds.) ICWE 2011. LNCS, vol. 6757, pp. 121–136. Springer, Heidelberg (2011)
2. Ceri, S., Fraternali, P., Bongio, A.: Web Modeling Language (WebML): a modeling language for designing Web sites. In: Procs of the WWW9 Conf., Amsterdam (May 2000)
3. Ferraiolo, D., Chandramouli, R., Kuhn, D.R.: Role-based access control, 2nd edn. Ebrary, INC., vol. xix, p. 381. Artech House, Boston (2007)
4. Ferrini, R., Bertino, E.: Supporting RBAC with XACML+OWL. In: Proceedings of the 14th ACM Symposium on Access Control Models and Technologies, SACMAT 2009, Stresa, Italy, June 03-05, pp. 145–154. ACM, New York (2009)
5. Finin, T., Joshi, A., Kagal, L., Niu, J., Sandhu, R., Winsborough, W., Thuraisingham, B.: Rowlbac: representing role based access control in OWL. In: Proceedings of the 13th ACM Symposium on Access Control Models and Technologies, SACMAT 2008, Estes Park, CO, USA, June 11-13, pp. 73–82. ACM, New York (2008)
6. Knechtel, M., Hladik, J.: RBAC authorization decision with DL reasoning. In: Proceedings of the IADIS International Conference WWW/Internet, pp. 169–176 (2008)
7. Koch, N., Kraus, A.: The Expressive Power of UML-based Web Engineering. In: Proceedings of the 2nd International Workshop on Web-Oriented Software Technology (IWOOST 2002), CYTED, pp. 105–119 (2002)
8. Lampson, B.W.: Dynamic Protection Structures. In: AFIPS Conference Proceedings, vol. 35 (1969)

9. Lima, F., Schwabe, D.: Application Modeling for the Semantic Web. In: Proceedings of LA-Web 2003, Santiago, Chile, pp. 93–102. IEEE Press (November 2003)
10. Schwabe, D., Rossi, G.: An object-oriented approach to Web-based application design. Theory and Practice of Object Systems (TAPOS), 207–225 (October 1998)
11. Vdovjak, R., Frasincar, F., Houben, G.J., Barna, P.: Engineering Semantic Web Information Systems in Hera. Journal of Web Engineering 2(1&2), 3–26 (2003)
12. Sandhu, R., Ferraiolo, D., Kuhn, R.: The NIST model for role-based access control: Towards a unified standard. In: Proceedings of the Fifth ACM Workshop on Role-Based Access Control, Berlin, pp. 47–63 (July 2000)
13. Berners-Lee, T., Connolly, D., Kagal, L., Hendler, J., Schraf, Y.: N3Logic: A Logical Framework for the World Wide Web. Journal of Theory and Practice of Logic Programming (TPLP), Special Issue on Logic Programming and the Web (2008)
14. Bonatti, P.A., De Coi, J.L., Olmedilla, D., Sauro, L.: Rule-Based Policy Representations and Reasoning. In: Bry, F., Małuszyński, J. (eds.) Semantic Techniques for the Web. LNCS, vol. 5500, pp. 201–232. Springer, Heidelberg (2009)
15. Mühleisen, H., Kost, M., Freytag, J.-C.: SWRL-based Access Policies for Linked Data. In: SPOT 2010 2nd Workshop on Trust and Privacy on the Social and Semantic Web, Heraklion, Greece (2010)
16. Hollenbach, J., Presbrey, J., Berners-Lee, T.: Using RDF Metadata To Enable Access Control on the Social Semantic Web. In: Workshop on Collaborative Construction, Management and Linking of Structured Knowledge (CK 2009) (ISWC 2009), Washington, DC (2009)