DESIGN AND RUN TIME REASONING WITH RELBAC

Rui Zhang, Bruno Crispo and Fausto Giunchiglia

November 2008

Technical Report # DISI-08-062
Design and Run Time Reasoning with RelBAC

Rui Zhang
DISI, University of Trento
Via Sommarive 14, 38100
Trento, ITALY
zhang@disi.unitn.it

Bruno Crispo
DISI, University of Trento
Via Sommarive 14, 38100
Trento, ITALY
crispo@disi.unitn.it

Fausto Giunchiglia
DISI, University of Trento
Via Sommarive 14, 38100
Trento, ITALY
fausto@disi.unitn.it

ABSTRACT
Relation-Based Access Control (RelBAC) is an access control model for the Web scenarios, which represents permissions as relations between users and objects. By exploiting the formalization of RelBAC model in Description Logics (DL), sophisticated access control policies can be directly encoded as DL formulas. This facilitates the administration with design time reasoning on hierarchies, memberships, propagations, separation of duties, etc. and helps with run time reasoning to make access control decisions. All these reasoning can be performed through state of the art, off-the-shelf DL reasoners.

General Terms
Access Control

Keywords
RelBAC, access control policies, design-time reasoning, run time reasoning

1. INTRODUCTION
Many new applications such as Online Social Networks, Blogs, Shared Desktops and more traditional applications redeveloped following the Software as a Service allow users to store, edit, share their data via the Web. The Web can potentially increase the number and the quality of user interactions with other users as a result of having a large community of connected users and shared data. Many of these interactions may span across several different organizations. They can be one-off, short term or long term interactions. These types of cross-organizational distribution pose new challenges for users, making it more difficult to share data in a protected way. As a consequence of that, many existing Web-based services fail to offer a flexible and fine-grained protection. In many cases, sharing is all-or-nothing while users need support for rich and fine-grained security policies to at least match those they can implement on traditional desktop applications. For many of these applications the data people may want to share, their protection and organization changes often and in an unpredictable way. This requires tools to support the owner of such data in specifying the access control rules and complex reasoning about policies, while the system is in operation.

In [12] we proposed a new access control model and a logic, called RelBAC (for Relation Based Access Control) which allows us to deal with such novel scenarios. The key idea, which differentiates the RelBAC model from the state of the art, is that permissions are modeled as relations between users (called subjects in access control terminology) and data (also called objects) while access control rules are their instantiations, with arity, on specific sets of users and objects. We define the RelBAC model as an Entity Relationship (ER) model [6] thus defining permissions as relations between classes of subjects and classes of objects. In this paper, we complete our model by describing its reasoning ability both at design and run time. By exploiting the well known translation of ER diagrams into Description Logics (DL) [2], we define a (Description) logic, called the RelBAC Logic, which allows us to express and reason about users, objects, permissions, access control rules and policies. In turn, this allows us to reason about policies by using state of the art, off-the-shelf, DL Reasoners, e.g., Pellet[20].

The rest of the paper is structured as follows. Section 2 shows a motivating example. Section 3 gives a brief introduction to RelBAC model. Reasoning with RelBAC for design time and run time is shown in Section 4 and Section 5. In Section 6 we'll show how RelBAC integrate a reasoner for reasoning tasks. Section 7 summaries related works and we conclude in Section 8.

2. MOTIVATING EXAMPLE
As a motivating example we use the case of a Web-based sales force automation (SFA) application. A SFA is typically a client-server application that provides a set of tools and services such as e-mail, reporting, database of contacts, a more or less complex document management system, to support salesmen in their pre-sale (i.e. preparing the offer) and post-sale (i.e. scouting for follow up contracts) activities. Most small and medium companies cannot afford to run their own server, so many vendors offer it also as a service, accessible by mean of a Web-based client. To be useful the service must offer to the management of the company a flexible and fine-grained access control able to map responsibilities and operations for the company’s sales force. At
the same time, each salesman, typically paid on bonuses, is quite protective about her own contacts and negotiations, so she should be able to indicate further security constraints to protect her own data. At the same time, any non trivial contract acquisition requires collaborations among salesmen and also support from the technical department to have any chance of success, so there should be also the possibility to specify temporary and dynamic access control policies.

In the above scenario we assume the following policies:

1. A sales agent can create at most one customer folder per sector.
2. At most 2 sales agents and a sales manager can be involved in a new offer provided they are not involved already in more than 3 other offers.
3. At least 2 sales agents should be involved in a sector.
4. Any sales agent cannot read more than 3 customer folders belonging to at most 2 different industrial sectors.
5. At least one sales agent with experience in Industrial Sector ICT must be involved in Offer Information Highways.

In the next section we introduce the RelBAC model and then illustrate through the paper how to apply it on our motivating example.

3. RelBAC
In this section, we describe briefly the RelBAC model that we proposed in [12] and its logical framework.

As is shown in the ER Diagram of Figure 1, RelBAC has the following components. SUBJECT (or USER): a subject is a user (or her agent) that requests an access to some resources. OBJECT: an object is any resource of the system a user requests access. PERMISSION: the intuition is that a PERMISSION is an operation that users can perform on objects. To capture this intuition a PERMISSION is named with the name of the operation it refers to, e.g., Write, Read operation or some more high-level operation, e.g., Assign or Manage. In RelBAC model, the original form of a verb is used as a PERMISSION name with the first letter capitalized.

For SUBJECT and OBJECT, the loops represent the IS-A relations that form the hierarchies of user groups and object classes. This is coherent to the tradition how people organize their desktop resources: a top-down tree-like file system. The most interesting part is the loop on PERMISSION which represents the IS-A relations among named pairs, e.g., Update(manager1, offer1), etc. such that the set theory applies on PERMISSIONs (sets of named pairs). For example, the IS-A relation that 'Update is more powerful than Read' can be captured with the intuition that those people who can access some data with the permission 'Update' should have already been assigned the permission 'Read'.

In RelBAC, a RULE associates a PERMISSION to a specific set of (SUBJECT,OBJECT) pairs. The Entity-Relationship model of RelBAC can be formalized directly in Description Logic [2] as listed in Table 1.

Now we show how to use them for our motivating example. We abstract the scenario into ER diagrams as Figure 2 and Figure 3. SUBJECT: employees, sales directors, sales managers, sales agents and sales representatives; OBJECT: digital documents for sales, sectors, customers, offers; PERMISSION: involve, manage, create, read, update, delete.

Using standard DL formulas, the policies of our motivating can be translated into the following RULEs:

1. Agent ⊑≤ 1Create.(Customer ⊓≤ 1Compose.Sector)
2. {Offer ⊑≤ 2Involve.Agent ⊓≤ 1Involve.Manager, Employee ⊑≤ 3Involve−1.Offer}
3. Sector ⊑≥ 2Involve.Agent
4. Agent ⊑≤ 3Involve−1.(Customer ⊓≤ 2Compose.Sector)
5. ≥ 1Involve.(Agent ∩ InvolvedIn : ICT)(ih)
CASE 1: ‘A concept should not be declared directly or indirectly as a subconcept of itself.’

According to the interpretation of RelBAC, the antisymmetry property of the partial order ‘⊆’ applies to groups, classes and permissions. To enforce this constraint, we exploit subsumption check as follows.

\[ \mathcal{P} \models C \sqsubseteq D? \]

For group hierarchy, given two groups \(U_1, U_2\), RelBAC checks the subsumption as \(\mathcal{P} \models U_1 \sqsubseteq U_2\)? A ‘Yes’ answer restricts \(U_2 \sqsubseteq U_1\) to be added to \(\mathcal{P}\). For example, a sales manager is also an employee can be formalized as ‘Manager \(\subseteq\) Employee’ and if the administrator asserts by mistake that an employee is also a sales manager, a check of ‘\(\mathcal{P}, S \models Manager \sqsubseteq Employee?\’ is processed and help avoid loops in the group hierarchy.

CASE 2: ‘An individual should not be declared belonging to a concept and its subconcept at the same time.’

Given that \(U_1 \sqsupseteq U_2\), for any user \(u\), if \(U_1(u)\) holds then we have \(U_2(u)\) implied by the reasoner. Thus this constraint can be rephrased as ‘a user should not be declared as member of a group she already belongs to.’ This constraint relates to two administration operations, add and delete which we will discuss in details in Section 4.2.

CASE 3: ‘A set cannot be subset of two sets that are mutually exclusive.’

This constraint holds according to the following theorem.

\[ \{U_1 \cap U_2 \subseteq \bot, U \subseteq U_1, U \subseteq U_2\} \models U \sqsubseteq \bot \]

For example, Manager \(\cap\) Agent \(\subseteq\) \(\bot\) formalizes that Sales Manager and Sales Agent are mutually exclusive. Then any attempt to assign one user to both groups will be checked as inconsistent. This is used also on separation of duties in Section 4.4.

Notice that not all paths in the hierarchies imply inheritance. We just model those inheritable with partial order and formalize them into subsumption formulas. Here we talked about user group ‘IS-A’ hierarchy only, and of course these cases can be applied to permission and object hierarchies according to classic set theory.

4.2 Membership

Employees of our SFA scenario are grouped as sales director, sales manager, etc. Similar to what RBAC does with roles, RelBAC provides an access control mechanism based on membership of groups such as to grant that sales managers can read offers. With the growing size and number of the groups, the management of user membership becomes crucial. The RelBAC logic can help the administrator to control these memberships.

Adding (deleting) an individual user from an existing subject group means only adding (deleting) an assertion to (from) the knowledge base \(S\). For example, to add a user \(u\) as a sales manager, we can just add to \(S\) one assertion \(Manager(u)\). This will give \(u\) all the permissions assigned to Manager just as the assignment of a role in RBAC. However, before adding this state assertion to \(S\), the administrator must check the following two properties:

**Redundancy** An assertion is redundant if it can be inferred from the existing knowledge base already. To check that \(u\)
As shown in Figure 4(a), the most complex situation is when membership of \( u \) to \( U_i \) is propagated from \( U_j \) which satisfies \( U_j \supseteq U_i \) with some intermediate group \( U''_k \). Thus the deletion of \( u \) from \( U_i \) requires the compensation of adding \( u \) as member of all \( U_k, U''_k \ldots \) till it reaches \( U_i \). As an example, let us assume that in our motivating example there is an extra group \( \text{PowerfulAgent} \) that enable members to be both a sales agent and a sales manager. So the resulting hierarchy is as Figure 4(b). If the administrator wants to remove this anomaly and make sure that no sales agent can be assigned as a sales manager then the deletion of membership to \( \text{Manager} \) requires not only the removal of assertion \( \text{PowerfulAgent}(john) \) but the compensation of adding assertion \( \text{Agent}(john) \). The process will consist in compensation to \( \text{Manager}(john) \) as in step 3 and deletion of this assertion in the next loop.

In this section we discussed only groups membership, however, object classes and permissions are also sets, with objects or (subject, object) pairs as elements, their membership management is then dealt similarly to that of groups.

### 4.3 Propagation

Membership and permission propagation are not new. In RBAC [8], user assignments and permission assignment propagate through role hierarchy. The senior roles inherit the permission from junior roles and junior roles inherit users from senior roles.

Sometimes, this propagation is not welcomed from management point of view. For example, a CEO does not have (and she does not want or need) all the permissions on all the operations a sales agent can perform. The CEO retains responsibility for all the employees of her company without the need to perform all the operations they can perform. We will show details how membership and permission propagate in the RelBAC model in this section.

#### 4.3.1 Membership Propagation

The advantage of RelBAC is that the natural model of ‘IS-A’ relation brings ‘free’ propagation through ‘IS-A’ hierarchies. By ‘free’ we mean that no extra rule is needed to specify the propagation after the ‘IS-A’ relations are clearly designed. That is to say, user membership propagation depends on group ‘IS-A’ hierarchy only. Given any two groups \( U_i, U_j \) such that \( U_j \supseteq U_i \) and \( u \) is a member of \( U_i \) then the membership of \( u \) to \( U_j \) can be automatically implied by the reasoner. Notice the transitivity of partial order is preserved by the model ‘\subseteq’ so it’s not necessary that \( U_i, U_j \) are directly connected in the hierarchy. Similarly, an object membership propagates through class ‘IS-A’ hierarchy.

For example, if Bob is a sales manager and Manager \( \geq \) Employee then Bob is an employee comes for free as the reasoner entails that \( \{ \text{Manager}(bob), \text{Manager} \subseteq \text{Employee} \} \models \text{Employee}(bob) \).

#### 4.3.2 Permission Propagation

The permission propagation is more complex because a RelBAC permission is a binary relation that links a subject to an object. Thus it has three paths to propagate: ‘IS-A’ hierarchy of subjects, objects and permissions.

1. A permission in RBAC is a pair \((p, o)\) where \( o \) is the object and \( p \) is the operation to access with. In RelBAC, a PERMISSION is a relation between SUBJECT and OBJECT.
Policies in form of \( U_i \subseteq U_j \) provide a way to build ‘IS-A’ hierarchy of user groups. Thus, permissions propagate from junior group to senior groups as

\[
\{ U_j \subseteq U_i, U_i \subseteq \alpha \} \models U_j \subseteq \alpha
\]

in which \( \alpha \) stands for some permission assignment. For example, \( \text{Manager} \geq \text{Employee} \) implies that all the permissions assigned to the employees propagate to sales managers.

In addition to the group hierarchy which simulates the role hierarchy in RBAC model, RelBAC provides object class and permission hierarchy with partial order \( \geq \) applied on classes and on permissions as \( \text{O}_i \subseteq \text{O}_j \) and \( \text{P}_i \subseteq \text{P}_j \).

For two assignments \( \beta, \beta' \) with the same permission \( P \), but on different object classes \( \text{O}_i, \text{O}_j \), if \( \text{O}_i \geq \text{O}_j \), the propagation goes in different ways according to the semantics of the assignment.

- If \( \beta, \beta' \) are assignments onto some \((\text{only}, \text{at least} n)\) objects in \( \text{O}_i \) and \( \text{O}_j \), then
  \[
  \{ \text{O}_i \subseteq \text{O}_j, U \subseteq \beta \} \models U \subseteq \beta'
  \]

For example, sales agents are allowed to read some \((\text{only}, \text{at least} 3)\) processed offers implies that sales agents are allowed to read some \((\text{only}, \text{at least} 3)\) offers because processed offers are subset of offers as in the assertion \( \text{Processed} \subseteq \text{Offer} \).

- If \( \beta, \beta' \) are assignments onto all \((\text{at most} n)\) objects in \( \text{O}_i \) and \( \text{O}_j \), then
  \[
  \{ \text{O}_i \subseteq \text{O}_j, U \subseteq \beta \} \models U \subseteq \beta'
  \]

For example, employees are allowed to read all \((\text{at most} 5)\) offers implies that sales managers are allowed to read all \((\text{at most} 5, \text{maybe less})\) processed offers because \( \text{Processed} \subseteq \text{Offer} \).

Permissions can propagate through the permission hierarchy as well. In contrast to sets of individuals such as groups or classes, the partial order among permissions describes subsumption between sets of \((u, o)\) pairs. For example, \( \text{Manage} \subseteq \text{Read} \) implies that any assignment with permission \( \text{Manage} \) is also assigned with \( \text{Read} \) such as those are allowed to manage offers implies that they are also allowed to read offers.

Propagations are the result of the reasoning on all the three kinds of ‘IS-A’ hierarchies together (through the partial order). No specific propagation rules are necessary for such propagations. This feature will simplify the system design and reduce the possibility of errors.

### 4.4 Separation of Duties

Advanced access control models support Separation of Duties (SoD) as an important security property. Here, we discuss SoD in general and the support of high level concerns about SoD as discussed in [17].

#### 4.4.1 Separation of Duties

In RelBAC, a permission is a relation that links a subject with an object, and for SoD the only thing to do is to assert axioms about permissions.

For example, two steps of a bank transaction are to initiate a transaction and to authorize the transaction. The two steps should be separated for clerk and supervisor of the bank. RelBAC can describe this SoD with an axiom as follows that can be checked by the reasoner.

\[
\text{Initiate} \sqcap \text{Authorize} \subseteq \bot
\]

In general, given \( n \) steps of a task as \( \text{step}_1, \ldots, \text{step}_n \), a SoD enforces at least \( k(2 \leq k \leq n) \) users take all these duties, which means that any user can have at most \( m = [n/(k-1)] - 1 \) of these duties. Thus any \( m + 1 \) of these rights should not be assigned to any single user. Then RelBAC can formalize this as follows.

\[
\bigcup_{i=1}^{[n/(k-1)]} (\cap_{j=1}^{[n/(k-1)]} U_{ij}) \subseteq \bot
\]

in which \( U_{ij} \) stands for the groups for each group at most \( m \) arbitrary duties are assigned.

Suppose in our example that the permission \( \text{Manage Offer} \) requires 3 steps \( \text{create offer, fulfill offer and archive offer} \) and an SoD enforces that at least 2 employees should be involved in managing an offer. This SoD can be enforced as follows.

\[
\exists \text{Create.Offer} \sqcap \exists \text{Fulfill.Offer} \sqcap \exists \text{Archive.Offer} \sqsubseteq \bot
\]

as \( C_{\bigcup}^{[n/(k-1)]} = C_{3}^{[[3/(2-1)]]} = C_{3} = 1 \).

#### 4.4.2 High Level Constraint of SoD

For general SoD constraints, the composition of \( k \) users to complete a task is sometimes important. N.Li et al. introduced an algebra in [17] to specify complex policies combining requirements on user attributes and cardinality. Apart from the cardinality for a given permission, their algebra can specify the composition of the users for the SoD which they regard as high-level policy. Examples of such high-level policies are the following:

1. Exactly two users, one sales manager and one sales agent.
2. At least one sales manager and one sales agent, and some other sales managers or agents.
3. At least one sales manager and one sales agent, and some other employees besides sales managers and agents.

RelBAC can express such constraints using object-centric rules as follows:

\[
\text{Offer} \sqsubseteq (= 2 \text{Manage}^{-1}.\text{User}) \sqcap (= 1 \text{Manage}^{-1}.\text{Manager}) \sqcap (= 1 \text{Manage}^{-1}.\text{Agent})
\]

\[
\text{Offer} \sqsubseteq (\forall \text{Manage}^{-1}.(\text{Manager} \sqcup \text{Agent})) \sqcap (\exists \text{Manage}^{-1}.\text{Manager}) \sqcap (\exists \text{Manage}^{-1}.\text{Agent})
\]

\[
\text{Offer} \sqsubseteq (\exists \text{Manage}^{-1}.\text{Manager}) \sqcap (\exists \text{Manage}^{-1}.\text{Agent})
\]

Here we abbreviate the usage of both \( \geq n \) and \( \leq n \) as \( = n \) in standard DL value restriction.

### 5. RUN TIME REASONING

Once we expressed the set of policies that apply to a system as a RelBAC knowledge base, run time reasoning can be performed for access control decision and dynamic separation of duties.
5.1 Access Control Decision

Basically, to decide whether a user \( u \) has some permission \( P \) on some object \( o \), \( RelBAC \) submit a query for knowledge \( P(u, o) \) to the knowledge base predefined with \( P, S \). If \( P(u, o) \) is inferred by the knowledge base, the decision should be ‘Yes’; otherwise, ‘No’. In addition to this, \( RelBAC \) is able to take decisions on many complex access requests as follows. Here user \( u \) belongs to a group \( U \), an object \( o \) belongs to a class \( O \), and \( P \) is a permission.

- Is \( u \) allowed to access \( o \) with \( P \)? For example, is a sales manager named Hill allowed to read an offer called ‘Server’?
  \[ P, S \models P(u, o) \text{? e.g., Read(hill, Server)} \]

- Is \( u \) allowed to access some objects in \( O \) with \( P \)? For example, is Hill allowed to read maximum 5 of the processed offers?
  \[ P, S \models \exists P.O(u) \text{? e.g., } \exists \text{Read.Processed}(hill) \]

- Is \( u \) allowed to access maximum/minimum \( n \) of the objects in \( O \) with \( P \)? For example, is Hill allowed to read maximum 5 of the processed offers?
  \[ P, S \models (\exists P.O(u)) \text{? e.g., } (\exists \text{Read.Processed}(hill)) \]

Here value restriction \( \leq n \) is used to express maximum \( n \), \( n=5 \). Other number restrictions such as minimum are straight forward. The exact restriction \( = n \) can be expressed with combination of maximum and minimum. Strictly more than \( n \) and less than \( n \) can be achieved with minimum \( n+1 \) and maximum \( n-1 \) because in \( RelBAC \), number restrictions are about natural number only.

- Is \( u \) allowed to access all the objects in \( O \) with \( P \)? For example, is Hill allowed to read all the processed offers?
  \[ P, S \models (\forall O.P)(u) \text{? e.g., } (\forall \text{Processed.Read}(hill)) \]

- Is there any user(s) in \( U \) allowed to access all objects in \( O \) with \( P \)? For example, is there any sales manager allowed to read all the processed offers?
  \[ P, S \models O \subseteq \exists P^{-1}.U \text{? e.g., } \exists \text{Processed}.\exists \text{Read}^{-1}.\text{Manager} \]

because the virtual group \( \exists \text{Read}^{-1}.\text{Manager} \) denotes the set of all the objects that can be read by some sales managers.

- Are there maximum/minimum \( n \) users in \( U \) allowed to access all objects in \( O \)? For example, is there minimum 3 managers allowed to read all the processed offers?
  \[ P, S \models O \subseteq (\geq)P^{-1}.U \text{? e.g., } \exists \text{Processed} \geq 3 \text{Read}^{-1}.\text{Manager} \]

- Is each user of \( U \) allowed to access maximum/minimum \( n \) objects in \( O \)? For example, is every sales manager allowed to read more than 10 offers?
  \[ P, S \models U \subseteq (\geq)nP.O \text{? e.g., } \text{Manager} \geq 11 \text{Offer.Read} \]

- Is each user of \( U \) allowed to access all objects in \( O \) with \( P \)? For example, is each of the managers allowed to read all the processed offers?
  \[ P, S \models U \subseteq \forall O.P \text{? e.g., } \text{Manager} \subseteq \forall \text{Offer.Read} \]

because the virtual class \( \forall \text{Offer.Read} \) denotes a set of all the users that can read all the processed offers.

5.2 Dynamic Separation of Duties

Separation of Duties (SoD) is categorized into static and dynamic. Static SoD can be represented in \( RelBAC \) as discussed in Section 4.4.1. Dynamic SoD intuitively allows the duties assigned to one user in case that they are not activated simultaneously at run time.

To enforce dynamic SoD, \( RelBAC \) introduces a new kind of permission, \( run time permissions (RTP) \) to describe the state of individuals at run time. A RTP is a permission describing the execution of a permission. For each permission in form of a verb (phrase), the corresponding RTP is the present continuous tense of the verb. For example, the RTP of permission ‘read’ is ‘reading’. To support dynamic SoD, for each \( RelBAC \) permission, a RTP is introduced. Moreover a user cannot have an RTP unless she has the original permission. For the example of permission ‘read’, Reading \( \geq \) Read is used to restrict that a user cannot execute permission ‘reading’ without the permission ‘read’. Thus the dynamic SoD ‘an offer cannot be read by some agent while being updated by a sales manager’ is specified as follows.

\[ \exists \text{Reading}^{-1}.\text{Agent} \cap \exists \text{Updating}^{-1}.\text{Manager} \subseteq \perp \]

Dynamic SoD is specified at design time and enforced at run time. The knowledge base must be informed of all activated permissions such as \( \text{Ann is reading an offer ‘Information Highway’} \) and be updated adding a new assertion \( \text{Reading(ann, ih)} \). Then the dynamic SoD will take effect that no manager can update offer ‘Information Highway’ which is an entailment of the reasoner that

\[ P, S \models \{ \text{Reading(ann, ih)} \} \models (\text{Updating : ih}) \cap \text{Manager} \subseteq \perp \]

If a request of a manager to update \( \text{ih} \) comes, it’ll be rejected.

6. AUTOMATED REASONING

In this section, we present the architecture of a system implementing the \( RelBAC \) model. As shown in Figure 5, the user interface (UI) stands between users and the other components of the system and interprets the queries to the knowledge base. The knowledge base consists of two parts namely, \( P \) and \( S \). The reasoner interacts with databases and users.
(through UI). Hollow arrows stand for user related operation or information exchange; solid arrows represents internal data flow and interaction.

- From the perspective of an administrator, the hollow arrows (a) and (b) are direct queries and updates to the knowledge base, where the solid arrow (h) represents the interaction of knowledge between \( P \) and \( S \). The reasoning service at design time to perform redundancy checking and conflict checking are offered to the administrator by arrow (c). Arrow (e), (g) and (f) stand for reasoning with \( P \), \( S \) and both, corresponding to standard TBox, ABox and TBox+ABox reasoning in DL.

- From a requester point of view, a query for permission is interpreted by the UI and handed to the reasoner through arrow (d). The reasoner processes the query as an entailment reasoning with respect to the knowledge base through arrows (e), (f) and (g) to provide access control decisions.

Our implementation integrates an open source reasoner Pellet [20] through owl-api. The state of the art Pellet 2.0 offers necessary reasoning ability for entailment, consistency, subsumption, and realization. The incremental reasoning about ABox updates in Pellet is useful for \( RelBAC \) as the run time membership update relies on ABox changes and reasoning.

To get a test data for the system, we build an ontology with RDF/OWL language describes the scenario of the motivating example. Sample policies listed in Section 2 are covered by these ontologies. We randomly generated for groups and classes hundreds of individuals. We test consistency checking performance on this knowledge base with the results listed in Table 2.

A small business with 5 user groups, 9 object classes, about 400 individuals (including employees and documents) with 10 access control rules can be formalized in an ontology of 61.0kb. And it takes less than 50 ms to complete consistency checking as is shown by the first record of the table. When the business grows in the number of set (either group or class) and in the number of rules into more than 10 times (as record 2), the checking time grows too. Even just increase the number of individuals randomly for users or objects, the time consumed grow exponentially. It is a preliminary test, but shows that the general purpose reasoner does not perform well on an access control problem of the Web scenario. Although \( RelBAC \) uses a decidable DL language, to achieve a real-time access control system still requires much work.

### 7. RELATED WORK

Access control is not a new topic. The amount of work which has been done on \( RBAC \) and its level of development is incomparably high (see, e.g., [8, 1, 19] or [4]). A preliminary version of \( RelBAC \) was introduced in [12]. In that paper we introduced the idea and the logic of the model but we did not cover the reasoning ability of the model, that are the main topic of this paper. As already hinted in the previous sections the main difference of \( RelBAC \) with respect to \( RBAC \) is that the former models permissions as ER relations thus making them first class objects (which can evolve independently of users and objects), and thus allowing for arity aware access control policies. Furthermore the use of ER relations allows for a direct embedding of policies into a (Description) Logic which allows to reason about them. Yet another difference from \( RBAC \) is the formation of hierarchies. Role hierarchies serves as an advanced feature in \( RBAC \) but not necessarily true for different scenarios as discussed by Li et al. in [16]. \( RelBAC \) provides not only the partial order for permission propagation with natural formalization as discussed in Section 4.3, but a way to formalize any binary relations that forms the hierarchy which doesn’t propagate permissions.

A lot of work has also been developed towards providing logical frameworks which would allow to reason about \( RBAC \) based policies, see, e.g., [3, 13, 18]. Besides the differences in the underlying logic and in the specifics of the formalizations, a conceptual difference is that all these logical frameworks have been added on top of \( RBAC \), while \( RelBAC \) is defined natively with its own (Description) Logic. As a non trivial plus of our approach, it becomes possible (with only a bit of effort) in \( RelBAC \) to have non-logic experts to handle policies and to reason about them using state of the art reasoning technologies (the SAT technology - used within DL reasoners - is by far the most advanced technology and the one mostly used in real world applications).

Some work has also been done in formalizing \( RBAC \) in DL. Thus, for instance, DL is used in [21] in order to formalize relations as binary roles while, more recently, Jung-Hwa Chae et.al use DL to formalize the object hierarchy of \( RBAC \) [5]. This work is again very different from ours as here DL is just another logic used to reason about \( RBAC \) instead of the logic designed to express \( (RelBAC) \) policies.

Other researchers have dealt with the problem we are interested in. Thus for instance, Juri et al. propose an access control solution for sharing semantic data across desktops [7]. They use a three dimensional access control matrix to represent fine-grained policies. We see a problem in that their solution does not seem to scale well since the matrix grows polynomially with the number of objects and of sets of users sharing such objects (as from above, \( RBAC \) like \( RBAC \) does not have this problem since it uses hierarchies to represent knowledge about users, objects and permissions.) Other authors have addressed the problem of access control in open and dynamic environments by adapting \( RBAC \). One such approach is [3].

A lot of research has been done to use logic for policy verification [14, 9, 15]. Just to mention a couple of examples. Organisation has been considered as an extension of role in ORBAC model [14]. Kalam et al. used a first-order logic based logic to formalize the model, which models the control problem with named triples. In contrast, we use Description Logic as the formalism which is a decidable subset of first order logic. K. Fisler et al. proposed a tool named Margrave [9] implemented with BDD at the back end. Our solution

<table>
<thead>
<tr>
<th>Size(kb)</th>
<th>Set</th>
<th>Rule</th>
<th>Individual</th>
<th>Time(ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>61.0</td>
<td>14</td>
<td>10</td>
<td>426</td>
</tr>
<tr>
<td>2</td>
<td>86.8</td>
<td>141</td>
<td>131</td>
<td>162</td>
</tr>
<tr>
<td>3</td>
<td>141.1</td>
<td>141</td>
<td>133</td>
<td>483</td>
</tr>
<tr>
<td>4</td>
<td>273.4</td>
<td>141</td>
<td>131</td>
<td>805</td>
</tr>
</tbody>
</table>
8. CONCLUSION

RelBAC models permissions as binary relations, a first class component. It allows to express many complex properties, and especially powerful in arity related policies. In this paper, we illustrated on advantage of RelBAC that is the ability to use off-the-shelf reasoners to reason about typical access control problems and properties. In this first evaluation we showed that many reasoning tasks are supported. However, state of the art reasoners are not specifically designed for RelBAC so the time consumed is hardly ‘real-time’ as it is too slow. As part of future work we will study how to improve efficiency. We would like also to test the reasoner against policies more complex than the one considered in this paper and possibly extend the class of security properties we test. Other direction of future work is to exploit Semantic Matching [11] to support the generation of permissions based on similarity.

9. REFERENCES


