Department of
Information Engineering
and Computer Science DISI

UNIVERSITY
OF TRENTO - Italy

DISI - Via Sommarive 14 - 38123 Povo - Trento (Italy)
http://www.disi.unitn.it

DESIGN-TIME AND RUN-TIME REASONING WITH RELBAC

Rui Zhang

October 2009

Technical Report # DISI-09-058

Design-time and Run-time Reasoning with RelBAC

ABSTRACT

Relation-Based Access Control (*RelBAC*) is an access control model for the Web scenarios, which represents permissions as relations between users and objects. It allows to express policies using cardinality and quantifiers and to specify separation of duties in the basic model rather than as an additional constraint. This paper shows that 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, propagations, separation of duties, etc. and helps with run-time reasoning to make access control decisions. All these reasonings can be automated and 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 Web applications such as Online Social Networks, Blogs, Shared Desktops and more traditional applications re-developed as Web-based services allow users to store, edit, share their data over Internet. 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. They may involve individual, groups, communities and combinations of these.

Many existing Web-based services however fail to offer a flexible and fine-grained protection. In many cases, sharing is all-or-nothing while users need support to at least match those security policies they can implement on traditional desktop applications. The complexity of such scenarios 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.

RelBAC (*Relation Based Access Control*) is a new access control model and a logic proposed by Giunchiglia et al. in [11] that can offer such a support. 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 permission assignments are their instantiations, with arity, on specific sets of users and data. The *RelBAC* model is defined as an *Entity Relationship (ER) model* [6] thus defining permissions as relations between classes of subjects and classes of objects. This paper completes the model by describing its reasoning ability both at design-time and at run-time.

By exploiting the well known translation of ER diagrams into *Description Logics (DL)*[1] [2], we define a (Description) logic, called the *RelBAC* Logic, which allows us to express and reason about subjects, objects and permissions. In turn, this allows us to reason about policies by using state of the art, off-the-shelf, DL Reasoners, e.g., Pellet[19].

The rest of the paper is structured as follows. Section 2 briefly introduces the *RelBAC* model with a motivating example. Design-time and run-time reasoning with *RelBAC* are shown in Section 3 and Section 4. Section 5 shows the architecture of a *RelBAC* system integrated with a DL reasoner and some test results. Section 6 summaries related work and we conclude in Section 7.

2. *RELBAC*

In this section, we describe briefly the model and the logic of *RelBAC* via a motivating example from web based sales force automation (SFA).

2.1 Motivating Example

An SFA application 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 presale (i.e. preparing the offer) and post-sale (i.e. scouting

[1]A decidable sub-set of First Order Logic (FOL).

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

2.2 Why *RelBAC*?

RelBAC allows to model and express access control policies and the related properties (e.g. separation of duty) simpler than with other existing AC models. In particular

- *RelBAC* supports cardinality. Quantifiers have been very successfully used in data bases, but in access control often policies are implicitly universally quantified. They are useful since using them we can express, for example, access control rules which states that *students should be able to use at least one PC*. We are stating that any student in principle could use all PCs but that what really matters is that she has access to one. And the above policy could be made stronger, using number restrictions, by saying that a student should have access to exactly one PC or, using the universal quantifier, by saying that students can use only PCs and that therefore, e.g., they cannot use personal assistants. Of course the same effect can be obtained in the existing models, e.g., RBAC, by checking these as constraints at run time.
- Using cardinality, important properties such Separation of Duty as defined by Li et al. [], can be easily expressed as an access control rule and reason about it (as shown in Section 3.4) rather than as an additional contraints to the basic model as in RBAC.
- RelBAC splits subjects from objects by defining permissions as relations. The role of users and objects is completely symmetric and one can symmetrically define user-centric (i.e all senior managers can write file F) or object-centric policies (i.e. File F can be read only by senior managers).

2.3 The *RelBAC* Model and Logic

As is shown by the ER Diagram in Figure 1, What distinguishes *RelBAC* from other access control models is `PERMISSION` in addition to the basic components such as `SUBJECT` and `OBJECT`. The intuition is that a `PERMISSION` is an operation that users (`SUBJECT`s) can perform on certain resources (`OBJECT`s). To capture this intuition a `PERMISSION` is named with the name of the operation it refers to, e.g., *Write*, and *Read* operation or some more high-level operation, e.g., *Assign* or *Manage*. In *RelBAC*, the original form of a verb is used as a `PERMISSION` name with the first letter capitalized. The *generalization* (loops) on each com-

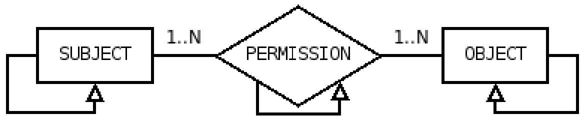

Figure 1: The ER Diagram of the *RelBAC* Model.

ponents represent `IS-A` relations. They are the most common and important relations among the knowledge. Groups of `SUBJECT` and classes of `OBJECT`[2] are organized with `IS-A` hierarchies. 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., $Update(ann, doc)$. Regarding a `PERMISSION` as a set of named pairs allows set theories exploited on `PERMISSION`s. For example, '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', which is represented as 'each subject-object pair named Update `IS-A` pair named with Read'. A `IS-A` relation can be represented as a subsumption axiom in *RelBAC* as

$$C \sqsubseteq D \ or \ P \sqsubseteq Q$$

where C and D are both groups or classes, and P and Q are both `PERMISSION`s.

In *RelBAC*, a `PERMISSION` assignment associates a `PERMISSION` to a specific set of (`SUBJECT`, `OBJECT`) pair(s) in the following forms of logic formulas.

$$\begin{array}{lllr} U \sqsubseteq \exists P.O & (1) & U \sqsubseteq \leq n\ P.O & (4) \\ U \sqsubseteq \forall P.O & (2) & U \sqsubseteq \forall \neg P.\neg O & (5) \\ U \sqsubseteq \geq n\ P.O & (3) & P(u,\ o) & (6) \end{array}$$

RelBAC is specially strong in representing *cardinality* in `PERMISSION` assignments. Formula 1 associates to `PERMISSION` P with the set of pairs formed with each individual of group U and *some (at least 1, maybe more)* instances in class O. Accordingly, Formulas 2-5 associate to P with

[2]Referred as group and class for short in the rest of the paper.

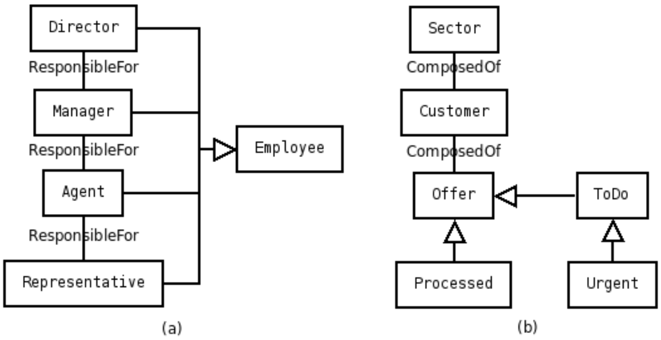

Figure 2: Subjects and Objects of an SFA Scenario.

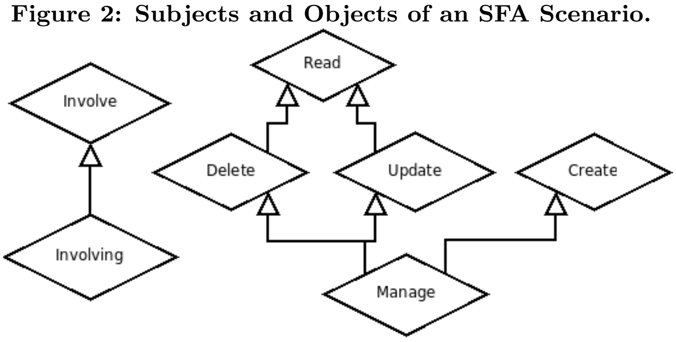

Figure 3: Permissions of an SFA Scenario.

the set of pairs formed with each individual of U and *only, minimum n, maximum n* and *all* objects in O. As for individuals, Formula 6 associates the pair (u, o) to P. The '*set*' constructor '$\{\cdot\}$' represents a set of individuals such as $\{u\}$. It is used to associate PERMISSION P with individuals such as $\{u\} \sqsubseteq \exists P.O$ with respect to Formula 1. Moreover, *RelBAC* can specify a PERMISSION assignment from the OBJECT perspective. By exchanging the places of U and O and replace P with its inversion P^-, Formula 1'-6' are specified, e.g., Formula 4' is $O \sqsubseteq \leq n\ P^-.U$, which specifies that 'each object in O can be access with P by maximum n subjects from U'.

Now we show how to apply *RelBAC* to the motivating example. Figure 2 models the subjects and objects hierarchies in which IS-A relation is modeled as *generalization* in contrast to other relations such as 'ResponsibleFor' and 'ComposedOf'. Figure 3 shows the relations among PERMISSIONs. With the formulas described above, the policies of in the motivating example can be written as the following PERMISSION assignments:

1. $Agent \sqsubseteq \leq 1 Create.(Customer \sqcap \leq 1 Compose.Sector)$
2. $Offer \sqsubseteq \leq 2 Involve.Agent \sqcup \leq 1 Involve.Manager$, $Employee \sqsubseteq \leq 3 Involve^-.Offer$
3. $Sector \sqsubseteq \geq 2 Involve.Agent$
4. $Agent \sqsubseteq \leq 3 Involve^-.(Customer \sqcap \leq 2 Compose.Sector$
5. $\{infoHighway\} \sqsubseteq \geq 1 Involve.(Agent \sqcap Involve^-.ICT)$

3. DESIGN-TIME REASONING

The benefit of expressing policies and security properties using *RelBAC* is the ability to reason about them. We can identify two phases of reasoning in *RelBAC*. At design-time, reasoning supports policy writers to detect possible redundancy and conflicts or to verify if the set of policies satisfy a desired static security property (i.e. separation of duties). The run-time reasoning abilities of *RelBAC* can be used to make access control decisions (coming in Section 4).

In *RelBAC*, the knowledge can be classified into two categories: policy and state. The knowledge dealing with domain terminologies of the access control model is called $\mathcal{P}$ (for Policy) such as the names of groups, classes, hierarchies, etc. The rest that relates to individuals is called $\mathcal{S}$ (for State) such as membership of a user to a group, an assignment to a individual user and/or object, etc. $\mathcal{P}$ is rather stable after design, while $\mathcal{S}$ is quite dynamic since it is sensitive to the system changes.

3.1 Hierarchy

Different relations between knowledge can form various hierarchies, even graphs such as shown in Figure 2. A feature of *RelBAC* is its natural formalization of the IS-A relation. For example, in Figure 2(b), the classification of offers into subsets *Processed, ToDo* and *Urgent* shows a hierarchy of IS-A relations among objects. Other relations such as *responsible for* or *composed of* can be formalized directly as binary relations. There are many practical constraints in hierarchy management and here we refer to group hierarchy as example and the theory applies also to hierarchies of OBJECT and PERMISSION. Two of the important constraints are considered to show the reasoning power of *RelBAC* for hierarchy management.

Constraint 1: '*A group should not be declared directly or indirectly as a sub-group of itself.*'
The representation of IS-A relation as subsumption axiom preserves the anti-symmetry property of the partial order. *RelBAC* provides *subsumption* check as follows.

$$\mathcal{P} \models C \sqsubseteq D?$$

For group hierarchy, given two groups U_1 and U_2, *RelBAC* checks whether the knowledge base infers that U_1 is subsumed by U_2 with the reasoning 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 manager is also an employee' can be formalized as '$Manager \sqsubseteq Employee$' and if the administrator asserts by mistake that 'an employee is also a manager', a check of '$\mathcal{P}, \mathcal{S} \models Manager \sqsubseteq Employee$?' is processed and help to avoid such mistakes.

Constraint 2: '*A group should not be declared as the subset of two sets that are mutually exclusive.*'
In *RelBAC*, *mutually exclusiveness* of two groups is represented as follows.

$$C \sqcap D \sqsubseteq \bot$$

Thus, the constraint is enforced in *RelBAC* with the following theorem.

$$\{U_1 \sqcap U_2 \sqsubseteq \bot, U \sqsubseteq U_1, U \sqsubseteq U_2\} \models U \sqsubseteq \bot$$

For example, $Manager \sqcap Agent \sqsubseteq \bot$ formalizes that 'Manager and Agent are mutually exclusive'. Then any attempt to assign a user to both groups can be avoided as it conflicts to existing knowledge base. This constraint is also useful in Section 3.4 when we discuss the property of Separation of Duties.

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 `IS-A` hierarchy of `SUBJECT` only, but these constraints are also valid for `OBJECT` and `PERMISSION` with similar reasoning.

3.2 Membership

Employees of our SFA scenario are grouped as director, 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 '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 group means only adding (deleting) an assertion to (from) the knowledge base $\mathcal{S}$. For example, to add a user u as a manager, we can just add to $\mathcal{S}$ 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 $\mathcal{S}$, 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 is a member of U_i is the *entailment* reasoning as follows:

$$\mathcal{P}, \mathcal{S} \models\ U_i(u)?$$

A 'Yes' answer means that this membership is not necessary because the knowledge base implies it already. This can happen in two cases. Either $U_i(u)$ exists already in $\mathcal{S}$ or u inherits the membership through group hierarchies.

Conflict If the knowledge base is consistent, but after adding the assertion it is no longer consistent any more, the assertion is conflicting with the knowledge base. To check conflict is the *consistency* reasoning as follows:

$$\mathcal{P}, \mathcal{S} \models \bot?$$

Conflicts are checked on the updated knowledge base $\mathcal{S}'$ which is $\mathcal{S} \cup \{U_i(u)\}$. A 'No' answer means that the updated knowledge base is still consistent and the operation to add the membership of u to U_i can be performed.

Referring to our example, adding *Hill* to the group of manager is equivalent to adding the assertion 'Hill is a manager' to S. This is achieved by following steps.

1. Redundancy checking with $\mathcal{P}, \mathcal{S} \models Manager(hill)$? if 'Yes', the assertion is redundant so no need to add it into the knowledge base.
2. Conflict checking with $\mathcal{P}, \mathcal{S} \models \neg Manager(hill)$? if 'Yes', the knowledge base implies that 'Hill is not a manager' and the assertion should not be added.
3. If the operation is neither redundant nor conflicting, add to the knowledge base $Manager(hill)$.

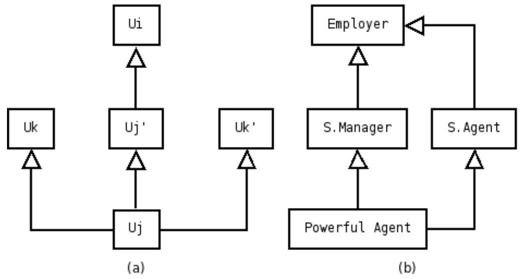

Figure 4: Delete Membership in Hierarchy.

When adding a user u as a member of a group U_i in a group hierarchy of partial order, we can do the same as without the hierarchy because the 'IS-A' relation does not bring exceptions for the redundancy and conflict checking. Deleting a user u from U_i, is more complicated considering that it might have impact on the membership of u to other groups in the hierarchy. In order to get the most specific group the user u belongs to, a form of *realization* reasoning is used to find the *most specific concept* U_i in the given concept set $\{U_1, ..., U_n\}$, such that $\mathcal{P}, \mathcal{S} \models U_i(u)$.

The following steps should be followed in order to delete a user membership in a group hierarchy.

1. Entailment checking $\mathcal{P}, \mathcal{S} \models U_i(u)$? A 'No' answer means that u is not a member of U_i and nothing else need to be done; otherwise go to Step 2.
2. Realization checking for the most specific group U_j u belongs and subsumption checking such that U_j satisfies $U_j \geq U_i$ (U_j can be exactly U_i).
3. Subsumption checking for all U_k such that $U_j \geq U_k$ and entailment checking such that $U_k(u)$ can be implied by the knowledge base.
4. Delete $U_j(u)$, and for all k add to knowledge base $U_k(u)$ after entailment checking for redundancy and conflict. And go to Step 1.

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 $U_j \geq U_i$ with some intermediate group U_j'. Thus the deletion of u from U_i requires the compensation of adding u as member of all U_k, U_k'... till it reaches U_i. As an example, let us assume that in our motivating example there is an extra group $PowerfulAgent$ that enable members to be both a agent and a sale manager. So the resulting hierarchy is as Figure 4(b). If the administrator wants to remove this anomaly and make sure that no agent can be assigned as a manager then the deletion of membership to $Manager$ requires not only the removal of assertion $PowerfulAgent(john)$ but the compensation of adding assertion $Agent(john)$. The process will consist in compensation to $Manager(john)$ 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.

3.3 Propagation

With the structure of knowledge defined, membership and access right may propagate through the relations. The natural formalization of IS-A relation in *RelBAC* leads to *free* propagations through IS-A hierarchies. By *free*, we mean that no extra effort is necessary except the specification of IS-A relations among knowledge. In this section, we will show how to manage such propagations with the help of design-time reasoning with *RelBAC*.

3.3.1 Membership Propagation

The membership of a SUBJECT u to a group U is a system state represented as '$U(u)$' in $\mathcal{S}$. Suppose there are two groups U_i and U_j such that U_i IS-A U_j, and u is a member of U_i. Then the fact that 'u is a member of U_j' can be inferred from the knowledge base which means the membership of u propagates from U_i to U_j. Notice that the transitivity of '$\sqsubseteq$' as a partial order, it is not necessary that U_i and U_j are directly connected with one IS-A relation in the hierarchy.

For example, if 'Bob is a member of the group $Manager$' and '$Manager$ IS-A *Employee*', then 'Bob is a member of the group *Employee*' comes for free as the knowledge base can infer

$$\{Manager(bob), Manager \sqsubseteq Employee\} \models Employee(bob)$$

Similarly, the membership of an OBJECT and a PERMISSION can propagate through the IS-A hierarchy of OBJECT and PERMISSION. This feature saves many membership assignments as they can be inferred by the knowledge base by reasoning.

3.3.2 Permission Propagation

The PERMISSION propagation is more complex than membership propagation, because in *RelBAC* a PERMISSION has three kinds of IS-A hierarchies to propagate i.e. the hierarchy of SUBJECT, of OBJECT and of PERMISSION itself.

The IS-A relation of SUBJECT are represented in *RelBAC* as *subsumption* axioms as '$U_i \sqsubseteq U_j$'. By the following reasoning

$$\{U_j \sqsubseteq U_i, U_i \sqsubseteq \alpha\} \models U_j \sqsubseteq \alpha$$

in which α stands for a permission assignment in Formula 1-5, a PERMISSION can propagate from a junior group to a senior group, e.g., $Manager \geq Employee$ implies that 'all the permissions assigned to the employees propagate to managers'.

For propagation through OBJECT hierarchies, given two assignments β and β' with the same permission P, but on different object classes O_i and O_j, if $O_i \sqsubseteq O_j$ the propagation goes in different directions according to the semantics of the assignment.

- If β and β' are P on *some / only / at least* n objects in O_i and O_j, then by reasoning as

  $$\{O_i \sqsubseteq O_j, U \sqsubseteq \beta\} \models U \sqsubseteq \beta'$$

  P propagates from O_i to O_j. For example, 'agents are allowed to read some / only / at least 3 processed offers' implies that 'agents are allowed to read some (only, at least 3) offers' because processed offers are subset of offers ($Processed \sqsubseteq Offer$).

- If β and β' are P on *all / at most* n objects in O_i and O_j, then by reasoning

  $$\{O_i \sqsubseteq O_j, U \sqsubseteq \beta'\} \models U \sqsubseteq \beta$$

  P propagates from O_j to O_i. For example, 'employees are allowed to read all / at most 5 offers' implies that 'managers are allowed to read all (at most 5) processed offers' because $Processed \sqsubseteq Offer$.

A PERMISSION can propagate through the IS-A hierarchy of PERMISSION as well. In contrast to sets of individuals such as groups or classes, the *subsumption* $P \sqsubseteq Q$ describes the IS-A relation between sets of (u, o) pairs. For example, $Manage \sqsubseteq Read$ implies that any assignment with permission $Manage$ is also assigned with $Read$ such as 'those are allowed to manage offers' implies that 'they are also allowed to read offers'.

Propagations of membership and PERMISSION through the three IS-A hierarchies are just reasoning result of the knowledge base without any policies regulating for these propagations. This feature will simplify the system design and reduce the possibility of errors.

3.4 Separation of Duties

RelBAC is very expressive because it models a PERMISSION as a binary relation between sets of SUBJECT and OBJECT, which allows the three components evolve relatively independently. Here we will show the rich expressiveness of *RelBAC* on representation of Separation of Duties (SoD) and furthermore, representation of the high-level constraints on the composition of the SUBJECTs in which the duties are distributed.

3.4.1 Separation of Duties

In general, given n duties of a task as d_1, ..., d_n, an SoD enforces that at least $k(2 \leq k \leq n)$ users should take all these duties. This means that any user can have at most m ($m = \lceil n/(k-1) \rceil - 1$) of these duties. Thus a single user should not be allowed to perform any $m+1$ of these duties. So *RelBAC* formalizes this SoD as follows.

$$\bigsqcup_{i=1}^{C_n^{\lceil n/(k-1) \rceil}} (\sqcap_{j=1}^{\lceil n/(k-1) \rceil} P_{ij}) \sqsubseteq \perp$$

in which P_{ij} stands for the jth PERMISSION out of the ith selected $m+1$ PERMISSIONs.

For example the task to manage an offer consists 3 duties as to create, to process and to archive the offer. An SoD enforces that 'at least 2 employees should be involved in managing an offer'. Suppose *Create*, *Process* and *Archive* are 3 permissions with co-domain as *Offer*, then this SoD can be represented as a policy as follows.

$$Create \sqcap Process \sqcap Archive \sqsubseteq \perp$$

As $C_n^{\lceil n/(k-1)\rceil} = C_3^{\lceil 3/(2-1)\rceil} = C_3^3 = 1$, there is only 1 conjunction of the 3 permissions in the disjunction axiom. The policy can be represented with mutually exclusive PERMISSIONs, which is more flexible than the mutually exclusive 'role's in $RBAC$ [8]. We can easily describe a duty with PERMISSION directly rather than create 'role's for each new duty.

3.4.2 High Level Constraint of SoD

For a general SoD, the composition of k users chosen to complete the task is sometimes important. Cardinality concern of the chosen SUBJECTs is a higher level constraint as the original SoD does nothing about it, e.g., to enforce that the set of SUBJECTs for the SoD above with the following composition.

1. Exactly two different users, i.e., one manager and one agent.
2. One manager and one agent, but the two can be the same.
3. At least one manager and at least one agent, and maybe some other managers or agents.
4. At least one manager and one agent, and some other employees besides managers and agents.

In $RelBAC$ such constraints are written as the following formulas:

1' $Offer \sqsubseteq (= 2Manage^{-}.User) \sqcap (= 1Manage^{-}.Manager) \sqcap (= 1Manage^{-}.Agent)$

2' $Offer \sqsubseteq (= 1Manage^{-}.Manager) \sqcap (= 1Manage^{-}.Agent)$

3' $Offer \sqsubseteq (\forall Manage^{-}.(Manager \sqcup Agent)) \sqcap (\exists Manage^{-}.Manager) \sqcap (\exists Manage^{-}.Agent)$

4' $Offer \sqsubseteq (\exists Manage^{-}.Manager) \sqcap (\exists Manage^{-}.Agent)$

Here we abbreviate both $\geq n$ and $\leq n$ as $= n$ in standard DL *number restriction*. N.Li et al. introduced an algebra in [16] to specify complex policies combining requirements on user cardinality. $RelBAC$ can capture all the concerns in their work with standard description logics.

4. 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.

4.1 Access Control Decision

$RelBAC$ the query for some fact γ can be answered by the following *entailment* reasoning.

$$\mathcal{P}, \mathcal{S} \models \gamma?$$

Basically, to decide whether a SUBJECT u has some PERMISSION P on some OBJECT o, $RelBAC$ queries $\gamma = P(u,\ o)$ to the knowledge base. If $P(u,\ o)$ is *entailed*, the decision should be 'Yes'; otherwise, 'No'. In addition to this, $RelBAC$ is able to take decisions on many complex access requests for each kind of permission assignment discussed in Section 2.3. Here user u belongs to a group U, an object o belongs to a class O, and P is a permission.

1. Is Hill allowed to read some processed offers? $\mathcal{P}, \mathcal{S} \models \{hill\} \sqsubseteq \exists Read.Processed$? (Formula 1)
2. Is Hill allowed to read more than 5 of the processed offers? $\mathcal{P}, \mathcal{S} \models \{hill\} \sqsubseteq \geq 5 Read.Processed$? (Formula 3) Number restriction '$\geq n$' formalizes the constraint of *minimum n*. The *maximum* constraint is straight forward by '$\leq n$'. The *exact* constraint '$= n$' can be formalized 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 $RelBAC$ 'n' is a natural number.
3. Is Hill allowed to read all the processed offers? $\mathcal{P}, \mathcal{S} \models \{hill\} \sqsubseteq \forall Processed.Read$? (Formula 5)
4. Is there any manager allowed to read all the processed offers? $\mathcal{P}, \mathcal{S} \models Processed \sqsubseteq \exists Read^{-}.Manager$? (Formula 1')
5. Are there at least 3 managers allowed to read all the processed offers? $\mathcal{P}, \mathcal{S} \models Processed \sqsubseteq \geq 3 Read^{-}.Manager$? (Formula 4')
6. Are urgent offers allowed to be read only by managers? $\mathcal{P}, \mathcal{S} \models Urgent \sqsubseteq \forall Read.Manager$? (Formula 2')

We can see from the above that flexible queries can be answered by the reasoner. So complex access control requests can be decided such as those requests with arity constraints.

4.2 Dynamic Separation of Duties

Separation of Duties (SoD) can be categorized into *static* SoD and *dynamic* SoD. A static SoD has been discussed in Section 3.4.1. A dynamic SoD enforces intuitively that the duties in a SoD are allowed to be assigned to one user at design-time, but not allowed to be activated simultaneously at run-time.

To enforce a dynamic SoD, $RelBAC$ introduces a new kind of permission, *run-time permission (RTP)*, to describe the state of permission execution at run-time. For a PERMISSION in form of a verb (phrase), the corresponding RTP is the present continuous participle of the verb. For example, the RTP of 'read' is 'reading'. To support dynamic SoD, for each 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 'reading' without assignment of 'read'. Thus the dynamic SoD 'an offer cannot be read and updated at the same time' is specified as follows.

$$Reading \sqcap Updating \sqsubseteq \bot$$

The knowledge base must update with all active PERMISSIONs. For example, 'an agent Ann is updating an offer $trento$' should be detected by the system monitor and $\mathcal{S}$ should be added a new assertion $Updating(ann, trento)$. Then the dynamic SoD will take effect that 'no one can update *trento*' as an *entailment* reasoning as follows.

$$\mathcal{P}, \mathcal{S} \sqcup \{Reading(ann, trento)\} \models \exists Updating.\{trento\} \sqcap Manager \sqsubseteq \bot$$

If a manager requests to update the offer *trento*, it will be rejected.

5. AUTOMATED REASONING

In this section, we present the architecture of a *RelBAC* system and some test results of the reasoner.

As shown in Figure 5, the user interface (UI) stands between users and the internal components: the knowledge base and the reasoner. The knowledge base consists of two parts: the policies $\mathcal{P}$ and the states $\mathcal{S}$. Hollow arrows stand for user related operation or information exchange; solid arrows represents internal data flow and interaction.

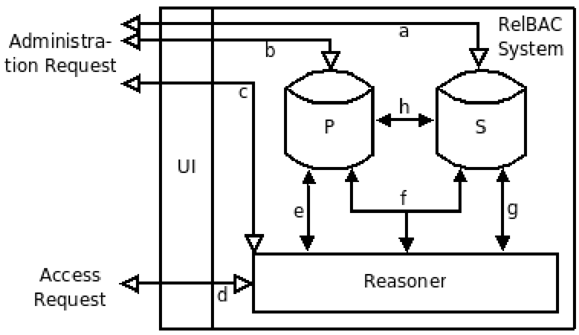

Figure 5: Architecture of a *RelBAC* System

	Size(kb)	Set	P.A.	Individual	Time(ms)
1	61.0	14	10	426	48.0
2	86.8	141	131	162	592.0
3	141.1	141	131	483	2191.0
4	273.4	141	131	805	5677.0

P.A.=Permission Assignment

Figure 6: Knowledge Base Consistency Test

- 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 $\mathcal{P}$ and $\mathcal{S}$. 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 $\mathcal{P}$, $\mathcal{S}$ 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 [19] through owl-api. A series of ontologies are built with RDF/OWL language about the scenario of the SFA example. Sample policies in Section 2.1 are covered by these ontologies together with randomly generated groups and classes plus hundreds of individuals.

A small business with 5 groups, 9 classes and about 400 individuals (including employees and documents) with 10 permission assignments (P.A.) is formalized in an ontology of 61.0kb. And it takes less than 50 ms to complete consistency checking (Record 1 of Figure 6). When the scale increases in the number of sets (either group or class) and P.A. for more than 10 times (as Record 2), the checking time also increases. In the remaining three case studies the numer of sets is increased of a factor of ten with the number of individuals ranging from roughly half the number of case 1 (162) to roughly double this number (805). The growth is relatively fast, as one would expect when dealing with DL reasoning problems. The good news, however, is that the overall times are within reason and close to real time (from 48 ms to 5677 ms). These first preliminary results are quite positive as we expect substantial speed ups when using dedicated AC focused reasoning heuristics.

6. RELATED WORK

As a well known access control model, the amount of work done on *RBAC* and its level of development are incomparably high (see, e.g., [8, 1, 18] or [4]). A preliminary version of *RelBAC* was introduced in [11]. In that paper we introduced the idea and the logic of the model but we did not cover the reasoning ability which is 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 resources), 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 [15]. *RelBAC* provides a way with the partial order for permission propagation in natural formalization (in Section 3.3), this can be also a way to formalize any binary relations that form the hierarchy (of users, objects and even permissions) which does not propagate permissions.

However, in our opinion, a much more interesting comparison between *RBAC* and *RelBAC* can be made by analyzing their similarities rather than their differences. The key observation is that *RelBAC* can be seen a *natural* extension of *RBAC* obtained exactly by adding what in the previous paragraph were listed as the main differences, namely arities in access control rules and the direct embedding of policies into a logic. On top of this, our preliminary experiments, which highlight a substantial similarity in the implementation (of ground policies) and in the user interactions, make us hope that in the end it will be possible to use *RelBAC* as as some kind of *enhanced RBAC*.

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, 12, 17]. 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 was used in [21] in order to formalize relations as binary roles while, more recently, J.Chae et al. used 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. Juri et al. proposed 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, *RelBAC*, 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].

Such research has also been done to use logic for policy verification [20, 13, 9, 14]. Just to mention some examples. Uszok et al. used DAML (http://www.daml.org) description logic based ontology for environments, contexts and policies in [20]. They model actions as DAML classes as well as subjects and objects, different from *RelBAC* in which permissions are formalized as DL roles. Organisation has been considered as an extension of role in *ORBAC*[13] in which 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 is based on Description Logic which is more expressive than a propositional logic. V. Kolovski et al. used Defeasible Description Logic rules to model the policy in [14]. However, the main difference between these approaches and *RelBAC* is not much about the reasoning abilities of one logic over another but rather the attempt made by *RelBAC* to provide a solution that allows to write access control rules using a well known notation (ER diagrams) and then translate them into DL. This has the clear advantage of using a well known methodology and the possibility to use a large variety of mature and well studied tools.

Li et al. present an algebra for high-level policy about complex SoD in [16]. It offers rich expressiveness for composition of a user set so as to enforce the Such expressiveness is covered by *RelBAC* without any extension.

7. CONCLUSION

RelBAC models permissions as binary relations which may evolve independently as a first class component. This allows to express many complex properties, and especially powerful in arity related policies. In this paper, we illustrated the advantage of *RelBAC* 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 part of the 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. Another direction of the future work is to exploit Semantic Matching [10] to support the generation of permissions assignments based on similarity.

8. REFERENCES

[1] A. Abou El Kalam, R. E. Baida, P. Balbiani, S. Benferhat, F. Cuppens, Y. Deswarte, A. Miège, C. Saurel, and G. Trouessin. Organization based access control. In *4th IEEE International Workshop on Policies for Distributed Systems and Networks (Policy'03)*, June 2003.

[2] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The description logic handbook: theory, implementation, and applications*. Cambridge University Press, New York, NY, USA, 2003.

[3] E. Bertino, B. Catania, E. Ferrari, and P. Perlasca. A logical framework for reasoning about access control models. *ACM Trans. Inf. Syst. Secur.*, 6(1):71–127, 2003.

[4] M. Bichler, J. Kalagnanam, K. Katircioglu, A. J. King, R. D. Lawrence, H. S. Lee, G. Y. Lin, and Y. Lu. Applications of flexible pricing in business-to-business electronic commerce. *IBM Systems Journal*, 41(2):287–302, 2002.

[5] J.-H. Chae and N. Shiri. Formalization of rbac policy with object class hierarchy. In E. Dawson and D. S. Wong, editors, *ISPEC*, volume 4464 of *Lecture Notes in Computer Science*, pages 162–176. Springer, 2007.

[6] P. P. Chen. The entity-relationship model - toward a unified view of data. *ACM Trans. Database Syst.*, 1(1):9–36, 1976.

[7] J. L. D. Coi, E. Ioannou, A. Koesling, and D. Olmedilla. Access control for sharing semantic data across desktops. In *1st International Workshop on Privacy Enforcement and Accountability with Semantics (PEAS)*, Busan, Korea, Nov. 2007.

[8] D. F. Ferraiolo, R. S. Sandhu, S. I. Gavrila, D. R. Kuhn, and R. Chandramouli. Proposed NIST standard for role-based access control. *Information*

and System Security, 4(3):224–274, 2001.

[9] K. Fisler, S. Krishnamurthi, L. A. Meyerovich, and M. C. Tschantz. Verification and change-impact analysis of access-control policies. In *ICSE '05: Proceedings of the 27th international conference on Software engineering*, pages 196–205, New York, NY, USA, 2005. ACM.

[10] F. Giunchiglia, M. Yatskevich, and P. Shvaiko. Semantic matching: Algorithms and implementation. *Journal on Data Semantics*, pages 1–38, 2007.

[11] F. Giunchiglia, R. Zhang, and B. Crispo. Relbac: Relation based access control. In *SKG '08: Proceedings of the 2008 Fourth International Conference on Semantics, Knowledge and Grid*, pages 3–11, Washington, DC, USA, 2008. IEEE Computer Society.

[12] S. Jajodia, P. Samarati, M. L. Sapino, and V. S. Subrahmanian. Flexible support for multiple access control policies. *Database Systems*, 26(2):214–260, 2001.

[13] A. A. E. Kalam, S. Benferhat, A. Miège, R. E. Baida, F. Cuppens, C. Saurel, P. Balbiani, Y. Deswarte, and G. Trouessin. Organization based access control. In *POLICY '03: Proceedings of the 4th IEEE International Workshop on Policies for Distributed Systems and Networks*, page 120, Washington, DC, USA, 2003. IEEE Computer Society.

[14] V. Kolovski, J. Hendler, and B. Parsia. Analyzing web access control policies. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pages 677–686, New York, NY, USA, 2007. ACM.

[15] N. Li, J.-W. Byun, and E. Bertino. A critique of the ansi standard on role-based access control. *IEEE Security and Privacy*, 5(6):41–49, 2007.

[16] N. Li and Q. Wang. Beyond separation of duty: an algebra for specifying high-level security policies. In *CCS '06: Proceedings of the 13th ACM conference on Computer and communications security*, pages 356–369, New York, NY, USA, 2006. ACM.

[17] F. Massacci. Reasoning about security: A logic and a decision method for role-based access control. In *ECSQARU-FAPR*, pages 421–435, 1997.

[18] M. J. Moyer and M. Ahamad. Generalized role-based access control. In *ICDCS*, pages 391–398, 2001.

[19] E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, and Y. Katz. Pellet: A practical owl-dl reasoner. *Submitted for publication to Journal of Web Semantics.*, 2003.

[20] A. Uszok, J. Bradshaw, R. Jeffers, N. Suri, P. Hayes, M. Breedy, L. Bunch, M. Johnson, S. Kulkarni, and J. Lott. Kaos policy and domain services: toward a description-logic approach to policy representation, deconfliction, and enforcement. *Policies for Distributed Systems and Networks, 2003. Proceedings. POLICY 2003. IEEE 4th International Workshop on*, pages 93–96, June 2003.

[21] C. Zhao, N. Heilili, S. Liu, and Z. Lin. Representation and reasoning on rbac: A description logic approach. In D. V. Hung and M. Wirsing, editors, *ICTAC*, volume 3722 of *Lecture Notes in Computer Science*, pages 381–393. Springer, 2005.