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

RELATION BASED ACCESS CONTROL: LOGIC AND POLICIES

Fausto Giunchiglia, Rui Zhang,
Bruno Crispo and Alessandro Artale

July 2010

Technical Report # DISI-10-053

Relation Based Access Control: Logic and Policies *

Fausto Giunchiglia[1,2], Rui Zhang[1], Bruno Crispo[2], and Alessandro Artale[3]

[1] CCST, Jilin University, Str. QianJin 2699, Changchun 130012, CHINA
[2] DISI, University of Trento, Via Sommarive 14, TN 38123, ITALY
[3] KRDB Research Centre, Free University of Bozen-Bolzano, ITALY

Abstract. The Web 2.0, GRID applications and more recently semantic desktop applications are bringing the Web to a situation where more and more data and metadata are shared and made available to large user groups. Things are further complicated by the highly unpredictable and autonomous dynamics of data, users, permissions and access control rules. For this novel scenario, a new access control model, Relation-Based Access Control (*RelBAC*) is proposed which allows subjects, objects and permissions to be defined independently. The key property which makes this possible is that permissions are modeled as relations between subjects and objects. *RelBAC* is formalized using the Description Logic $\mathcal{ALCQIBO}$, which allows to perform policy management, e.g., Separation of Duties via automated reasoning.

1 Introduction

Web service applications, GRID applications, the Web 2.0 and Social Web applications, e.g., FaceBook, MySpace, and more recently, semantic desktops (e.g., IRIS [17], Haystack [16], Nepomuk [26]) are bringing the Web to a situation where more and more user data and metadata are made available for sharing. In this context metadata may be tags, attributes of files, or complex graph structures such as file system or web directories, or (lightweight) ontologies[13]. In turn, users (actually user descriptions) can themselves be tagged by certain properties, they can be organized in groups, e.g., as the friends of a person, or as those people who are interested in a specific topic, e.g., "Peace in the Middle East", or in the results of a specific scientific experiment. Groups themselves can build complex graph structures (e.g., lightweight people ontologies written in FOAF), often across and independently of organizational boundaries, and also independently of how data and metadata are organized. This situation is further complicated by the high unpredictable dynamics where data, users, and access permissions change independently.

This new scenario presents a set of characteristics which make it radically different from previous applications, e.g., Intranet applications, in particular:

* This paper is a much extended version of the work originally published in [14] and extended in [15].

- Data and users are organized in complex structures; typically hierarchical structures, i.e., direct acyclic graphs (DAGs) plus constraints and complex links among user groups and data. Permissions themselves are organized in DAGs, needed to take into account, among other things, the time-variance of the application context [32, 31]. Thus, as an example, one would like to distinguish in a uniform way (and to reason about it, see item below) about *Read*, *Read during the week-end*, and *Read at night*.
- Permissions, and access control policies evolve autonomously from data and users. This requires treating them as first class objects. As a consequence, it must be possible to add, delete or change a permission or a rule independently from users and data (differently from what happens in *Role Based Access Control (RBAC)* [11]). Furthermore a much more refined control on the arity of access control rules must be supported. In particular, it must be possible to say that, e.g., m users in a pre-existing group can access n data in a newly created class.
- Systems are inherently open and it is impossible to know, at design time, the future evolution of data, metadata, users, user groups and the consequent access control policies. Data, metadata, users and user groups are subject to strong unpredictable dynamics. This requires complex reasoning about policies, at run time, while the system is in operation.

This paper proposes a new access control model, called *RelBAC* (for *Relation Based Access Control*) which allows us to deal with this novel scenario. The key idea, which differentiates *RelBAC* from the state of the art, is that permissions are modeled as relations between users and data (called *subjects* and *objects* in access control terminology), while access control rules are their instantiations, with arity, on specific sets of subjects and objects. The *RelBAC* model is defined as an *Entity Relationship (ER) model* [9] thus defining permissions as relations between classes of subjects and classes of objects. Finally, by exploiting the well known translation of ER diagrams into *Description Logics (DL)* [3] a domain specific (Description) logic, $\mathcal{ALCQIBO}$, is used to express and reason about subjects, objects, permissions and access control rules. In turn, this allows us to reason about policies by using state of the art, off-the-shelf, DL Reasoners, e.g., Pellet [30]. Thus, for instance the permission *Use* can be modelled as a binary relation which holds for all the pairs $< subj, obj >$ where the subject *subj* can *Use* the object *obj*, while $Student \sqsubseteq \exists Use.PC$ is an access control rule which states that all students should have access to at least one PC among all the PCs which are available. Furthermore, the policy consisting of the above control rule plus the rule $Student \sqsubseteq \leq 1 Use.PC$ states that students can use one and most one PC among all available PCs Notice that the above rule states that at different time points the same student may be allowed to use different PCs.

The rest of the paper is structured as follows. Section 2 describes the *RelBAC* model. Section 3 describes the Description Logic of $\mathcal{ALCQIBO}$. In Section 4, *RelBAC* is formalized with $\mathcal{ALCQIBO}$, together with the definition of hierarchies, general and ground access control rules and the most common access control rules TAC. Section 6 shows how to use *RelBAC*. Section 5 shows the

way to handle a popular security property, separation of duties with *RelBAC*. Section 7 describes the related work and Section 8 provides some conclusions.

2 The *RelBAC* Model

The *RelBAC* Model is represented with the ER Diagram as Figure 1. The details of the components are listed below.

Fig. 1. The ER Diagram of the *RelBAC* Model.

- `SUBJECT` (or `USER`): a set of subjects(e.g., a person or even a thread), namely the set of those users who can perform operations on objects; a set of subjects forms a `GROUP`. The loop on `SUBJECT` represents the 'IS-A' relation between sets of subjects. The largest `GROUP` is the collection of all the possible subjects.
- `OBJECT`: a set of all objects(e.g., anything with a URI), namely the set of all data (and/ or resources) on which users can perform operations; a set of objects forms a `CLASS`. The loop on `OBJECT` represents again an 'IS-A' relation between sets of objects. The largest `CLASS` is the collection of all the possible objects of the system.
- `PERMISSION`: intuitively, a permission is an operation that a subject is allowed to perform on an object. To capture this, a permission is named with the name of the operation it refers to (e.g., *Write* or *Download*). As shown in the ER diagram of Figure 1, a `PERMISSION` is a *relation* between `SUBJECT` and `OBJECT`, namely a set of (subject,object) pairs (e.g., Download(rui, /article1.pdf) or Read(fausto, http://disi.unitn.it/ knowdive/)). The loop on `PERMISSION` represents the 'IS-A' relation between sets of permissions. A Permission can be expressed both in term of subjects (i.e., students can use all PCs in room 29), and in term of objects (PCs in room 29 can be used by all students).
- `RULE` (short for `ACCESS CONTROL RULE`): a rule associates a `PERMISSION` to a specific set of `(SUBJECT,OBJECT)` pairs. A rule assigns the specific `SUBJECT` the access right to perform the operation named by the `PERMISSION` to the specific `OBJECT`. Rules are formalized as DL formulas, as described in the following subsection.

As an example of *RelBAC* model, consider the situation in Figure 2 where Rui is a person who has on his PC a directory of contacts (on the left) which represents his social network and a file system (on the right) which contains

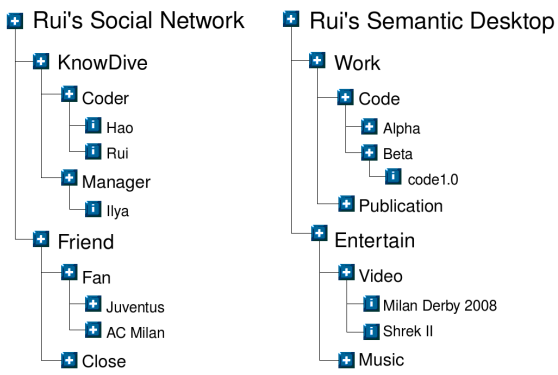

Fig. 2. SUBJECT and OBJECT Hierarchies.

various items including code, publications and entertainment material. In Figure 2, nodes with mark '+' represent a set (of people, of data), while nodes with mark 'i' represent a single entity (a person or a file). Thus, for instance, as from the picture, Rui has a colleague Hao, who's a coder in KnowDive research group. Rui has also another colleague, Ilya, who's a manager in the group; he has some friends who are soccer fans, some of which like the team AC Milan while some others like Juventus (another Italian team). He also has some close friends classified in his social network.

Consider now Figure 3. Permissions form a complex hierarchical structure. Similarly to Figure 2, nodes with mark '+' represent sets (of subject/ object pairs), while nodes with mark 'i' represent single subject/ object pairs. The hierarchy on the left in Figure 3 states that the *Read* permission is more general that the *Write* and *Delete* permissions, in other words, that having a *Write* or a *Delete* permission implies having also a *Read* permission. It also states (last item) that Hao can *Read Shrek II*, without necessarily being able to *Write* or *Delete* it.

Consider now the hierarchy on the right in Figure 3. This hierarchy shows how it is possible to represent contextual factors as direct conditions in a hierarchy. It states, for instance, that the users who can connect on weekdays are a subset of those who can have connect capability, and the same for those who can connect on weekends. Notice that in this hierarchy the root *Connect* is less general than its descendants and so on for all nodes and paths. In particular the people that will always be able to *Connect* will be a subset of those who will be able to Connect on week days or on weekends. The two hierarchies in Figure 3 have therefore opposite polarity, starting respectively from the relation with the largest and the smallest extension. The arrows, by going from the largest to the smallest relation, represent just this fact.

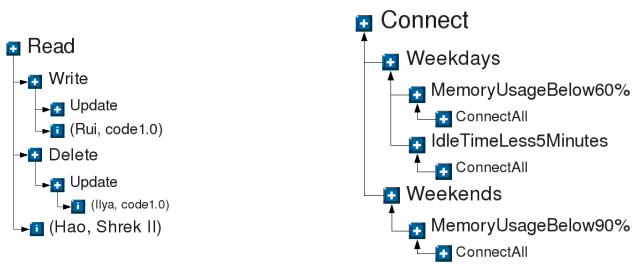

Fig. 3. Permission Hierarchies.

The ER Diagram modeling (a part of) the situation in the Figures 2, 3 also providing the missing information is depicted in Figure 4.[4] As it can be noticed, *Read* is defined between *Knowdive* and *Work* and used to define a many-to-many access control rule; all the other permissions define one-to-one rules, and *Update* is less general of both *Write* and *Delete*.

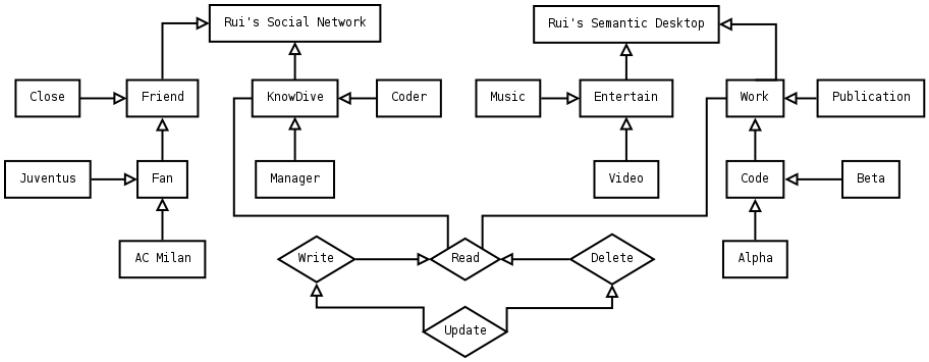

Fig. 4. A portion of the ER Diagram of Figures 2,3.

3 The Description Logic $\mathcal{ALCQIBO}$

The logic $\mathcal{ALCQIBO}$ extends the description logic $\mathcal{ALC}$ [3] with qualified cardinalities, inverse roles, nominals and Booleans for roles (see [29, 23, 22] for extensions of DLs with Booleans between roles). We define the syntax of $\mathcal{ALCQIBO}$ as follows.

Let $\mathsf{N_C}$, $\mathsf{N_R}$ and $\mathsf{N_I}$ be pairwise disjoint and countably infinite sets of *concept names*, *role names* and *individual names*. Then *concept expressions* and *role*

[4] To simplify the picture, as it is sometimes done, we draw the aggregation box only around the permission.

expressions are defined as follows:

$$\begin{aligned} C, D &::= A \mid \neg C \mid C \sqcap D \mid \geq n\, R.C \mid \{a_i\} \\ R, S &::= P \mid R^- \mid \neg R \mid R \sqcap S \end{aligned}$$

where $A \in \mathsf{N_C}$, $P \in \mathsf{N_R}$, $a_i \in \mathsf{N_I}$ and $n \in \mathbb{N}$.
A Knowledge Base (KB) is a pair $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ where $\mathcal{T}$, called *TBox*, is a finite set of *general concept inclusions (GCIs)* of the form $C \sqsubseteq D$ and a finite set of *general role inclusions (GRIs)* of the form $R \sqsubseteq S$, while $\mathcal{A}$, called *ABox*, is a finite set of concept and role assertions of the form $C(a_i)$ and $R(a_i, a_j)$, with $a_i, a_j \in \mathsf{N_I}$.

An $\mathcal{ALCQIBO}$*-interpretation*, $\mathcal{I}$, is a pair $(\Delta, \cdot^{\mathcal{I}})$ where Δ is a non-empty set called the *domain* of $\mathcal{I}$ and $\cdot^{\mathcal{I}}$ is a function mapping each $A \in \mathsf{N_C}$ to a subset $A^{\mathcal{I}} \subseteq \Delta$ and each $P \in \mathsf{N_R}$ to a relation $P^{\mathcal{I}} \subseteq \Delta \times \Delta$. Furthermore, $\cdot^{\mathcal{I}}$ applies also to individuals by mapping each individual name $a_i \in \mathsf{N_I}$ into an element $a_i^{\mathcal{I}} \in \Delta$ such that $a_i^{\mathcal{I}} \neq a_j^{\mathcal{I}}$, for all $i \neq j$, i.e., we adopt the so called *unique name assumption* (UNA). We extend the mapping $\cdot^{\mathcal{I}}$ to complex roles and concepts as follows:

$$\begin{aligned} (R^-)^{\mathcal{I}} &:= \{(y, x) \in \Delta \times \Delta \mid (x, y) \in R^{\mathcal{I}}\}, \\ (\neg R)^{\mathcal{I}} &:= \Delta \times \Delta \setminus R^{\mathcal{I}}, \quad (\neg C)^{\mathcal{I}} := \Delta \setminus C^{\mathcal{I}}, \\ (R \sqcap S)^{\mathcal{I}} &:= R^{\mathcal{I}} \cap S^{\mathcal{I}}, \quad (C \sqcap D)^{\mathcal{I}} := C^{\mathcal{I}} \cap D^{\mathcal{I}}, \\ (\geqslant n\ R.C)^{\mathcal{I}} &:= \{x \in \Delta \mid \sharp\{y \in \Delta \mid (x, y) \in R^{\mathcal{I}} \text{ and } y \in C^{\mathcal{I}}\} \geq n\}, \quad \{a_i\}^{\mathcal{I}} := \{a_i^{\mathcal{I}}\}. \end{aligned}$$

An $\mathcal{ALCQIBO}$-interpretation $\mathcal{I} = (\Delta, \cdot^{\mathcal{I}})$ is said a *model* of a KB, $\mathcal{K}$, iff it satisfies $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$, for all $C \sqsubseteq D \in \mathcal{K}$, $R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$, for all $R \sqsubseteq S \in \mathcal{K}$, $a_i^{\mathcal{I}} \in C^{\mathcal{I}}$, for all $C(a_i) \in \mathcal{A}$, and $(a_i^{\mathcal{I}}, a_j^{\mathcal{I}}) \in R^{\mathcal{I}}$, for all $R(a_i, a_j) \in \mathcal{A}$. In this case we say that $\mathcal{K}$ is satisfiable and write $\mathcal{I} \models \mathcal{K}$. A concept C (role R) is *satisfiable w.r.t.* $\mathcal{K}$ if there exists a model $\mathcal{I}$ of $\mathcal{K}$ such that $C^{\mathcal{I}} \neq \emptyset$ ($R^{\mathcal{I}} \neq \emptyset$).

As usual, we can define a number of useful abbreviations:

$$\begin{aligned} C \sqcup D &\text{ for } \neg(\neg C \sqcap \neg D) \\ (\leqslant n\ R.C) &\text{ for } \neg(\geqslant n+1\ R.C) \\ (= n\ R.C) &\text{ for } \neg(\geqslant n+1\ R.C) \sqcap (\geqslant n\ R.C) \\ \exists R.C &\text{ for } (\geqslant 1\ R.C) \\ \forall R.C &\text{ for } (\leqslant 0\ R.\neg C) \\ \top &\text{ for } A \sqcup \neg A \text{ (for some concept } A) \\ \bot &\text{ for } \neg\top \\ \mathcal{U} &\text{ for } R \sqcup \neg R \text{ (for some role } R) \end{aligned}$$

Note that, TBox axioms can be internalized. Indeed, we can encode each axiom $C \sqsubseteq D$ as the concept expression $\forall \mathcal{U}.(\neg C \sqcup D)$, while each role axiom $R \sqsubseteq S$ can be encoded as the concept expression $\forall \mathcal{U}.\forall(R \sqcap \neg S).\bot$. (To encode ABox assertions as concept expressions see [29]).

Concerning the complexity of $\mathcal{ALCQIBO}$, KB satisfiability can be reduced to reason over the two-variable first-order fragment with counting quantifiers

which is NExpTime-complete [28]. On the other hand, Boolean modal logic is a proper sub-language of $\mathcal{ALCQIBO}$ and it is NExpTime-complete [22]. As a consequence, reasoning in $\mathcal{ALCQIBO}$ is NExpTime-complete.

4 Formalization of *RelBAC*

The possibility of translating ER diagrams into Description Logics allows for a direct formalization of the *RelBAC* model into a family of logics for access control, more specifically, $\mathcal{ALCQIBO}$. In the following of this section we describe how it is possible to define user groups, object classes and permissions (Section 4.1), we show how it is possible to build hierarchies of users, objects and permissions (Section 4.2), and then show how to formalize general access control rules (Section 4.3) and instance specific access control rules (Section 4.4). Finally we conclude this section by showing how it is possible to formalize in *RelBAC* the type of access control rules used in *RBAC*) (Section 4.5).

4.1 Defining Subjects, Objects, Permissions

Sets of subjects and objects are formalized as atomic concepts. Permissions are formalized as DL roles (not to be confused with the *RBAC* roles!):

$$\begin{aligned} S_1, ..., S_m &\mid \text{(subjects)} \\ O_1, ..., O_n &\mid \text{(objects)} \\ P_1, ..., P_s &\mid \text{(permissions)} \end{aligned}$$

where $S_i (i = 1, ..., m)$ are concepts for subjects[5], such as $Friend$ or $KnowDive$; $O_j (j = 1, ..., n)$ are concepts for objects, such as $Video$ or $Code$; $P_k (k = 1, ..., s)$ are roles for permissions defining user-object pairs. Examples of permissions are conventional file operations such as $Read$ and $Write$ or some other field functions such as $Cash$ and $Audit$. $User$ and $Object$ are the concepts for all users and objects, respectively. From now on, we use *italic* starting with a Capital letter as concept and role names.

RelBAC provides ways to build complex groups and classes, and even complex relation based on these concepts and roles defined above through the logic of $\mathcal{ALCQIBO}$ with different constructors such as set operators, cardinality operators as described in Section 3.

4.2 Defining Hierarchies of users, objects and permissions

In $RelBAC$, we declare hierarchies as subsumption axioms, namely as axions of the form (we limit ourselves only to one direction of subsumption, dual arguments hold for the other direction):

$$A_i \sqsubseteq A_j$$

[5] $Subjects$ are also called $users$ in access control. The two terms are used interchangeably in this paper.

where A_i, A_j can be users, objects or permissions, whose meaning is

$$A_i^{\mathcal{I}} \subseteq A_j^{\mathcal{I}}$$

Thus for instance, some paths in the hierarchies in Figures 2, 3, are axiomatized as follows

$$Coder \sqsubseteq KnowDive$$
$$Code \sqsubseteq Work$$
$$Write \sqsubseteq Read$$
$$Connect \sqsubseteq Weekends$$

This allows to define partial orders $\geq$ on users, objects, and permissions, as follows. A `USER HIERARCHY` (represented in Figure 1 as the `IS-A` relation on `SUBJECT` and `GROUP`) is formalized as

$$U_1 \geq U_2 \text{ iff } U_1 \sqsubseteq U_2$$

A `OBJECT HIERARCHY` (represented in Figure 1 as the `IS-A` relation on `OBJECT` and `CLASS`) is formalized as

$$O_1 \geq O_2 \text{ iff } O_1 \sqsubseteq O_2$$

A `PERMISSION HIERARCHY` (represented in Figure 1 as the IS-A relation on the aggregation of `PERMISSION`, `SUBJECT` and `OBJECT`) is formalized as

$$P_1 \geq P_2 \text{ iff } P_1 \sqsubseteq P_2$$

Notice that the direction of the partial order on users and objects is opposite to that of subsumption; namely the smaller a set is, the higher it is in the partial order. $\geq$ has been defined, for users in particular, to mimic the $RBAC$ partial order on roles[11]. Notice however that the two partial orders are radically different, the first being a partial order on users, the second on permissions. The intuition is that the larger sets of users will have less permissions, and the same for objects. Similarly, notice that the partial order on permissions has the same direction as that of $RBAC$. This definition is coherent with $RBAC$: the more powerful permission the higher in the partial order. The overall intuition is that smaller sets of users and objects and therefore higher in their partial order are those involved in the more powerful permissions.

4.3 Defining General Access Control Rules

Access control rules may take one of the following three forms

$$C \equiv D$$
$$C \sqsubseteq D$$
$$C \sqsupseteq D$$

where: $C \equiv D$, to be read as "C is equivalent to D", is interpreted as $C^{\mathcal{I}} = D^{\mathcal{I}}$, C is either a group of users or a group of objects, and D can be any formula constructed following the syntax in Section 3. Equivalence should be used with a lot of attention and limited to those cases which are self-evident (e.g., synonyms) such as $ICTStudent \equiv ICTPeople \sqcap Student$. Rules usually take the form of subsumption formulas. In the following we will consider only one direction of subsumption; dual arguments apply for the other direction.

A first set of paradigmatic examples can be defined as follows:

$$U \sqsubseteq \exists P.O \tag{1}$$
$$O \sqsubseteq \exists P^{-1}.U \tag{2}$$
$$U \sqsubseteq \forall P.O \tag{3}$$
$$O \sqsubseteq \forall P^{-1}.U \tag{4}$$

For example, we can write:

- $Friend \sqsubseteq \exists Download.Music$ to state that all close friends can download some music,
- $Music \sqsubseteq \exists Download^{-1}.Friend$ to state that all music can be downloaded by some friend,
- $Friend \sqsubseteq \forall Download.Music$ to state that friends can download only music,
- $Code \sqsubseteq \forall Read^{-1}.KnowDive$ to state that the code can be read only by KnowDive members.

In RelBAC we can express cardinality in the rules as follows:

$$U \sqsubseteq \geq nP.O \tag{5}$$
$$O \sqsubseteq \geq nP^{-1}.U \tag{6}$$
$$U \sqsubseteq \leq nP.O \tag{7}$$
$$O \sqsubseteq \leq nP^{-1}.U \tag{8}$$

For example, we can write:

- $KnowDive \sqsubseteq \geq 1\ Program.Code$ to state that each KnowDive member should program for at least one project code,
- $Code \sqsubseteq \leq 2\ Program^{-1}.KnowDive$ to state that each project code should be programmed by at most 2 KnowDive members,

As it can be easily noticed from the examples above, and as also represented in the ER model in Figure 1, there are two kinds of access control rules

1. *User-centric access control rules* (e.g., Rule (1)), namely rules which define which users can perform an operation P on a certain set of objects;
2. *Object-centric access control rules* (e.g., Rule (2)), namely rules which define which objects can be applied a certain operation P^{-1} by a certain set of users.

In the above notation we have assumed that all permissions are defined from subjects to objects and therefore all object-centric rules are defined in terms on inverse permissions. Of course, in practice, the policy manager is free to define permissions as she wants.

4.4 Defining Ground Access Control Rules

Most often one would like to define *Ground Access Control Rules* (and policies), that we also sometimes call *Rules involving Instances*, namely statements about permissions of specific users and/or objects. In turn ground rules may involve *individuals* or *sets of individuals*.

Rules involving *individuals* (users of objects) can take one of two forms

$$\begin{aligned} U(u), O(o) \mid & \text{ (group or class assignment)} \\ P(u,o), P^{-1}(o,u) \mid & \text{ (permission assignment)} \end{aligned}$$

with the following intended (formal) semantics (the cases not considered are analogous):

$$\begin{aligned} (U(u))^{\mathcal{I}} &= u^{\mathcal{I}} \in U^{\mathcal{I}} \\ (P(u,o))^{\mathcal{I}} &= (u,o)^{\mathcal{I}} \in P^{\mathcal{I}} \end{aligned}$$

The first associates a user to a group while the second assigns a permission to a specific user and a specific object. We have the following examples:

- $KnowDive(Rui)$ declares Rui to be a member of the group $KnowDive$,
- $Video(Shrek\ II)$ states that the file $Shrek\ II$ is a video
- $Download(Hao, Shrek\_II)$ states that Hao can download $Shrek\ II$;

To define ground rules about *sets of individuals* we need some notation for sets and for computing the domain of a permission. We have the following:

$$\begin{aligned} \{a_1, \ldots, a_2\} \mid & \text{ (set constructor)} \\ P : o, P^{-1} : u \mid & \text{ (fill constructor)} \\ (P : o)(u), (P^{-1} : u)(o) \mid & \text{ (membership constructor)} \end{aligned}$$

where a_i can be a user or an object and the membership constructor is the composition of the fill constructor and the user assignment constructor, with the following semantics (the cases with the inverse permission are analogous):

$$\begin{aligned} \{a_1, \ldots, a_2\}^{\mathcal{I}} &= \{a_1^{\mathcal{I}}, \ldots, a_2^{\mathcal{I}}\} \\ (P : o)^{\mathcal{I}} &= \{u \in User^{\mathcal{I}} | (o,u) \in P^{\mathcal{I}}\} \\ ((P : o)(u))^{\mathcal{I}} &= u^{\mathcal{I}} \in (P : o)^{\mathcal{I}} \end{aligned}$$

Notice that the fill constructor $P : o$ defines all users u which have permission P on object o while the membership constructor states that the user u has

permission P on the object o. The above definitions allow us to define ground rules as follows:

$$U \sqsubseteq P : o \tag{9}$$
$$(P : o)(u) \tag{10}$$
$$(\exists P.O)(u) \tag{11}$$
$$(\forall P.O)(u) \tag{12}$$
$$(\geq nP.O)(u) \tag{13}$$
$$(\leq nP.O)(u) \tag{14}$$

For example, we can write:

- $CloseFriend \sqsubseteq Download : Shrek\ II$ to state that close friends can download $Shrek\ II$,
- $CloseFriend \sqcap \neg\{Hao\} \sqsubseteq Download : Shrek\ II$ to state that Hao is an exception to the previous policy;
- $\{Hao, Ilya\} \sqsubseteq Download : Shrek\_II$ to enumerate the two close friends who can download $Shrek\ II$,
- $(Download : Shrek\_II)(Hao)$ to state that Hao can download $Shrek\ II$,
- $\exists Update.Beta(Hao)$ to state that Hao can update at least a file of beta code,
- $\forall Upload.Video(Hao)$ to state that Hao can upload only video,
- $\geq 10 Update.Beta(hao)$ to state that Hao can update at least ten files of Beta code,
- $\leq 15 Download.Video(Hao)$ to state that Hao can download at most fifteen videos.

Policies (15–20) can be symmetrically stated for objects using inverse permissions, thus obtaining the following:

$$O \sqsubseteq P^{-1} : u \tag{15}$$
$$(P^{-1} : u)(o) \tag{16}$$
$$(\exists P^{-1}.U)(o) \tag{17}$$
$$(\forall P^{-1}.U)(o) \tag{18}$$
$$(\geq nP^{-1}.U)(o) \tag{19}$$
$$(\leq nP^{-1}.U)(o) \tag{20}$$

4.5 Defining the Total Access Control Rule

The usual practice in access control, and specifically in $RBAC$, is to construct policies as follows. First Roles are defined as sets of permissions P over a specific set of objects O; let us call this set, $P(O)$. Then $P(O)$ is assigned to individual users u or sets of users U. This assignment is *total* in the sense that *all the users in* U, or u herself in the case of single user, can apply each permission in P to *all*

objects in O. We call this the *Total Access Control (TAC) rule*. The *TAC* rule can be defined as the following *RelBAC* ground policy:

$$\{P(u_1, o_1), \ldots, P(u_1, o_m), \ldots, P(u_n, o_m)\}.$$

In other words the *TAC* rule is defined as the set of all ground access control rules $P(u_i, o_j)$ for all users $u_i \in U$ and objects $o_j \in O$.

A more elegant formulation defines the *TAC* rule as a policy which does not need to enumerate all instances. The DL formula for the *TAC* rule[6] is defined as follows:

$$\forall O.P \equiv \forall \neg P.\neg O$$

We have in fact that

$$\begin{aligned}(\forall \neg P.\neg O)^{\mathcal{I}} &= \{u \in User^{\mathcal{I}} |\ \forall o.\ (u, o) \in \neg P^{\mathcal{I}} \rightarrow o \in \neg O^{\mathcal{I}}\} \\ &= \{u \in User^{\mathcal{I}} |\ \forall o.\ o \in O^{\mathcal{I}} \rightarrow (u, o) \in P^{\mathcal{I}}\} \\ &= \forall O.P\end{aligned}$$

We can therefore define a *TAC* general policy using one of the following DL formulas:

$$U \sqsubseteq \forall O.P \tag{21}$$

$$O \sqsubseteq \forall U.P^{-1} \tag{22}$$

and similarly in the case of ground TAC policies:

$$(\forall O.P)(u) \tag{23}$$

$$(\forall U.P^{-1})(o) \tag{24}$$

We have the following examples:

- $Manager \sqsubseteq \forall Code.Update$ states that all project managers can update all the code,
- $Code \sqsubseteq \forall Manager.Update^{-1}$ states that all the code can be updated by all project leaders

It is interesting to notice that the two examples above of the *TAC* rule, namely

$$\begin{aligned}Manager &\equiv \forall Code.Update \\ Code &\equiv \forall Manager.Update^{-1}\end{aligned}$$

are equivalent in the sense that they define exactly the same set of policies. This is due to the fact that the *TAC* rule (and its strong request to be able to access all objects) plus the equivalence relation break the asymmetry intrinsic in subsumption and in user versus object centric access control policies (on this last item see below Section 6.3).

[6] Alex Borgida, borgida@cs.rutgers.edu

As a follow-up on the last observation, it can be noticed that the *RelBAC* *TAC* rule which does exactly the same as the rules used in *RBAC* is defined as follows:

$$U \equiv \forall O.P \tag{25}$$

or, from above, equivalently as the following formula [7].

$$O \equiv \forall U.P^{-1}, \tag{26}$$

5 Separation of Duties

Separation of Duties (*SoD*) is an important security property in modern access control systems[20]. It enforces that more than one person is required to complete a task. In this section, we will discuss the general meaning of *SoD*, its enforcement at design and run time and a *high-level* security policy [21] for *SoD*.

5.1 General *SoD*

An *SoD* states that, if a sensitive task consists of two steps, then two different users should perform mutually exclusive steps. More generally, when a sensitive task is composed of n steps, an *SoD* constraint requires the cooperation of at least k (for some $k \leqslant n$) different users to complete the task.

In one of the most well-known access control model, Role Based Access Control (RBAC)[11], *SoD* is enforced with the help of restrictions on the 'roles[8]'. The *SoD* that 'different users should perform two different steps of a task' can be enforced at design time by restricting any user from the assignments to the two roles, each of which is assigned the permission to carry out a step of the two. For a more general *SoD* that 'different users should perform n different steps of a task', each step is modeled as an *RBAC*role R_i, and the *SoD* is formally expressed as the following formula: $S_1 \sqcap \ ... \sqcap S_i \sqcap \ ... \ \sqcap S_n \sqsubseteq \bot$ where S_i is a set of subjects.

In *RelBAC*, a permission is a relation which links a subject with an object. To enforce this *SoD* in *RelBAC*, we need to assert an axiom which allows permissions. For example, suppose in a scenario of sales force automation[9], *to initiate, process, check and archive an order should not be completed by only one user.* Suppose *Initiate, Process, Check* and *Archive* are four permissions with the same domain as users and co-domain as orders. The *SoD* above can be expressed in *RelBAC* as

$$Initiate \sqcap Process \sqcap Check \sqcap Archive \sqsubseteq \bot$$

[7] See Section 6.2 on the use of equivalences in the definition of access control rules.

[8] RBAC roles should not be misused with DL roles, a 'role' is a component simulating a real world group, e.g., a set of managers. A user can only execute a permission assigned to the role that s/he can activate.

[9] http://www.salesforce.com

This policy restricts any pair (u, o) from belonging to all four sets *Initiate, Process, Check* and *Archive*.

In general, given n steps of a task $step_1, ..., step_n$, the *SoD* requires at least k ($k \leqslant n$) users can fulfill all these n steps. Suppose any of the k users can fulfill maximum m steps. In the worst case, everyone can fulfill equal number of steps, and satisfies the following in-equations:

$$(k-1) * m < n \quad i.e. \quad m \leqslant \lceil n/(k-1) \rceil - 1 \tag{27}$$

because $k,\ m,\ n$ are all integers. This means intuitively that any user can be assigned to at most m of these duties as restricted in Formula 27. Thus, any $m+1$ of these duties should not be assigned to one user. Then *RelBAC* can enforce the *SoD* with the following axiom:

$$\bigsqcup_{i=1}^{C_n^{\lceil n/(k-1) \rceil}} \left(\bigsqcap_{j=1}^{\lceil n/(k-1) \rceil} P_{ij} \right) \sqsubseteq \bot \tag{28}$$

in which P_{ij} stands for one of the m permissions for each step.

Following the example above, given the 4 duties in the SFA scenario, an *SoD* requires that *at least 3 users should be involved*. This *SoD* can be enforced as follows.

$$(Create \sqcap Update) \sqcup (Review \sqcap Create) \sqcup (Update \sqcap Archive) \sqcup$$
$$(Update \sqcap Review) \sqcup (Archive \sqcap Create) \sqcup (Review \sqcap Archive) \sqsubseteq \bot$$

as $C_n^{\lceil n/(k-1) \rceil} = C_4^{\lceil 4/2 \rceil} = C_4^2 = 6$.

Up to now, we have discussed the *SoD* designed by the administrator off-line. An *SoD* can be enforced both at design and at run time. An *SoD* enforced at run time intuitively means that the duties to be separated cannot be executed by her simultaneously no matter these duties are assigned to her or not off-line.

RBAC enforces *SoD* at run time with an extension of a component called *session*. In this extended model, if a user wants to activate roles, she has to activate a session first. Then, the session activates the roles in respect of her. A run-time *SoD*, which is called *Dynamic Separation of Duties (DSD)* in *RBAC*, enforces the session of a user is not allowed to activate roles in the *DSD*, no matter such roles are assigned to her at design time or not.

In *RelBAC*, we do not need an extension. The use of a special kind of permission, which we call *Run Time Permission (RTP)*, solves this problem. Assume that the name of a permission in the original *RelBAC* model is an English verb (or a verb phrase with a major verb describing the operation). Its corresponding *RTP* is formed by alternating the verb with the present participle of the verb. A permission's *RTP* describes its current execution. From the semantics of a permission defined in *RelBAC*, a user cannot be executing a permission unless she has been assigned it. This can be guaranteed by the following axiom.

$$Doing \sqsubseteq Do \tag{29}$$

where *Do* stands for a permission in the original *RelBAC*. Thus, to enforce a user is not allowed to have some permissions at run time, is to avoid the assignment of their *RTP*s to the user. For example, assume that *Initiating* is the *RTP* for the permission *Initiate*. As said above, a user cannot execute the permission *Initiating* without having the permission *Initiate*. Now, the *SoD* '*a user cannot initiate and process an order at the same time*' is enforced as follows (we assume that both permissions have the same range, *Order*):

$$Initiating \sqcap Processing \sqsubseteq \bot$$

In the real world, a user can be granted the permission to initiate an order (as a customer) and to process an order (as a sales agent), but cannot perform the two permissions (duties) simultaneously in order to avoid the process of one's own order.

Although an *SoD* can be enforced at design and run time with similar role axioms in *RelBAC*, the mechanisms regulating it are different. To enforce an *SoD* at run time, the monitoring mechanism should inform the access control system in real time, so that the current state such as *Alice is initiating an order 'Bolzano'* should be recognized. Then the knowledge base should be updated with the new assertion $Initiating(alice, bolzano)$. Therefore, the above *SoD* (ruling out the possibility of initiating and processing an order at the same time) in the KB $\mathcal{K}$ entails the following:

$$\mathcal{K} \sqcup \{Initiating(alice, bolzano)\} \models \neg Processing(alice, bolzano)$$

although Alice might have permissions to both initiate and process some order.

5.2 High Level Security Policy about *SoD*

SoD has been studied in [21] on detailed requirements for specific users in addition to numbers of each kind of users. An algebra was proposed to specify complex policies combining requirements on user attributes and number. On top of the cardinality constraints for given duties, the algebra can specify the composition of the users for the *SoD* which they regard as *high-level* security policy. For example, beyond to restrict that at least 2 users should be involved for the 4 steps in the *SFA* schema of Section 5.1, it further enforces that the set of users that can complete the order fulfillment task involve customer(s) to initiate the order and sales manager(s) to check the order. For example, the composition of the user set should be as follows.

1. At least one sales manager has to check orders and at least one customer to initiate orders.
2. At least one sales manager and at least one customer and maybe some other sales manager or customer are involved, but no others than those two kinds of users.
3. Exactly two users are involved, one sales manager and one customer.

Case 1 means that a customer should be involved to initiate the order and a manager should be involved to check the order, for the other users, the administrator does not care who processes and archives the order. Case 2 specifies that only customer and manager can be involved which means the same manager (or some other manager) will take charge of the duties to process and to archive. Case 3 is the most strict by allowing only one customer and one manager to be involved.

RelBAC can achieve this kind of constraints with *object-centric* rules with the *cardinality restriction* constructor. For example, as for the cases above, the three constraints for the set of users can be formalized as follows.

$$\begin{aligned} Order &\sqsubseteq (\geqslant 1\ Initiate^{-}.Customer) \sqcap (\geqslant 1\ Check^{-}.Manager) \\ Order &\sqsubseteq \forall Involve.(Customer \sqcup Manager) \sqcap \\ &\quad (\geqslant 1\ Initiate^{-}.Customer) \sqcap (\geqslant 1\ Check^{-}.Manager) \\ Order &\sqsubseteq (= 2\ Involve.(Customer \sqcup Manager)) \sqcap \\ &\quad (= 1\ Initiate^{-}.Customer) \sqcap (= 1\ Check^{-}.Manager) \end{aligned}$$

Where *Involve* is a permission more general than any of the 4 permissions for the 4 duties in the above schema, i.e.

$$Initiate^{-} \sqcup Process^{-} \sqcup Check^{-} \sqcup Archive^{-} \sqsubseteq Involve$$

This kind of *high-level* security policy complements the general *SoD* policy as discussed in Section 5.1 because it has the full power to describe the composition of the set of subjects including the exact cardinality.

6 Using *RelBAC*

How should we use *RelBAC* in practice? Isn't *RelBAC* just the *(n+1)* Logic for access control? More precisely, how can we use the added expressibility of *RelBAC* policies? This is still too early to judge and a lot of work has to be done in order to provide an answer to this question and, ultimately, to judge the real usefulness of *RelBAC*. However a few comments and observations can already be done. Let us analyze them in some detail.

6.1 Quantifiers in policies

The first observation is concerned with the role of quantifiers. Why should they be used at all? They have been very successfully used in data bases, but in access control they are completely avoided as policies are implicitly universally quantified (see Section 3). Do we really need them? Maybe access control relations do not need the level of expressibility needed in data bases and information systems.

Let us consider, as an example, the following access control rule

$$Student \sqsubseteq \exists Use.PC$$

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 (as in the TAC rule) 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 systems, e.g., $RBAC$, by checking these constraints at run time. But in this case this constraint would be embedded in the code and it would be impossible to reason about it. Notice that ER diagrams have been invented just for providing high level semi-formal specifications of information systems and Description Logics have been defined in order to perform automated reasoning about their properties. Maybe, in the past, there was no much need of high level specifications of the kind allowed by ER diagrams and even less of reasoning about them. But the increasing number of open, dynamically evolving systems, with strong access control requirements, which are among the main motivations for this work, seem to lead in this direction.

6.2 Subsumption policies

In state of the art access control systems, policies are stated as equivalences. In other words, in any moment in time, a given set of users is given exactly a set of permissions on a precisely defined set of objects. In $RelBAC$ we have suggested to minimize equivalences and to concentrate instead on subsumption policies (Section 4.3). This suggestion is a consequence of the past experience which has shown that stating properties (in our case policies) as equivalences leads into specifications which are too rigid, hard to maintain and that can easily create difficulties (e.g., generate an inconsistent set of policies). And this is more and more true the more complex systems are, and the more dynamics there are (with the need, each time a policy is changed, to check that all desired properties are satisfied).

Consider for instance the following access control rule:

$$Student \equiv \exists Use.PC$$

Suppose that, by chance, it happens that students may also use a smart-phone. One would like to add the following policy

$$Student \sqsubseteq \exists Use.Smart - phone$$

but this would lead an inconsistent theory (under the assumption that smart-phones are different objects from PCs), while this would have not been the case if we had used the corresponding subsumption policy, as written in the previous subsection. Dually, it is possible to assert the following rule

$$Student \sqsupseteq \exists Use.HPC$$

where HPC is an acronym for *High performance Computer*, and add later further policies which restrict the extension of *Student* just to the correct set of students.

Again, as in the previous subsection, similar effects could be obtained programmatically by dynamically controlling the extensions of the relevant sets of users and objects but, again, this would make it impossible to reason about them at the policy level. The further (usual) advantage of stating a policy in a logical specification, instead of embedding it into the code, is that it can be (easily) changed, contrarily to the latter case where the policy is hardwired in the system code.

6.3 User and Object centric policies

Prior to the success of $RBAC$ and the most recent access models for access control, this task was done by using *Access Control Matrix* [6]. A main advantage of this approach was that the Access Control Matrix could be analyzed by rows or by columns. By looking at the rows one would take the users' perspective and analyze their *capabilities*, by looking at the columns and one would take the objects' perspective and analyze their *access control lists*. One main problem was scalability: in large applications the large number of subject/ object pairs, most of which were irrelevant, made this approach unfeasible in practice. $RBAC$ solved this problem by splitting subjects from objects via roles. This however leads to a user centric view of policies where the key component is the definition of $RBAC$user roles.

Instead, *RelBAC* splits subjects from objects by defining permissions as relations. As the previous sections make clear, the role of users and objects is completely symmetric and one can symmetrically define user-centric or object-centric policies. In practice, the policy administrator can look at (our version of) capabilities or at (our version of) access control lists. It is important to notice that in the Web we find more and more applications, e.g., Wikipedia, various content portals, where the space of users is quite flat (i.e., most of the users are undistinguished users, often anonymous, which navigate the Web) while data form a huge space of valuable content whose access needs to be put more and more in control (think of instance of the sensitive topics, e.g., sex).

6.4 Scalability

But, will *RelBAC* scale in practice? This issue is fundamental not only because the current state of the art, e.g., $RBAC$, has been vey successful on this issue, but also because the new full connectivity scenarios are bringing us to applications where the size of users and data is far beyond the existing applications.

The answer to this question must be split in two parts: ground policies and general policies. According to our first implementation of $RelBAC$, ground policies in $RelBAC$ can be implemented, using, e.g., a relational data base, by using practically the same ideas as $RBAC$, and with very much the same level of efficiency. In practice the triples $\langle S, O, P\rangle$ implementing $RelBAC$ access control rules can be implemented as pairs $\langle S, P(O)\rangle$, very much in the same way as the

rules used in *RBAC*. Also the *RelBAC* policy maintenance problem is basically the same and the system administrator can be provided an interface which looks very much the same as in *RBAC*.

Things change radically at the level of general policies. Here there are many concurrent issues. The first is the number of policies. On this issue things look promising. In fact even if *RelBAC* policies are inherently more expressive, they extend naturally one of the fundamental features which made *RBAC* very successful, i.e, the hierarchy on roles and the propagation of permissions, to users, objects and permissions (see Section 4.3), which in turn leads to the possibility of generation in *RelBAC* of what we could call *Hierarchical Policies*. Consider for instance a policy of kind (1) from Section 4.3 (the same argument applies also to all the other policies):

$$U_1 \sqsubseteq \exists P_1.O_1.$$

This policy also implies the following set of policies

$$U_2 \sqsubseteq \exists P_2.O_2.$$

for any U_2 such that $U_2 \geq U_1$, for any O_2 such that $O_1 \geq O_2$, and for any P_2 such that $P_2 \geq P_1$. In other words, the number of subsumption policies can be minimized by taking the biggest possible group of users, the smallest possible set of objects and the most powerful permission. All policies involving any subgroup, any superset of objects and any less powerful policy are automatically implied. As a simple example based on the hierarchies in Figures 2,3, consider the following policy

$$Knowdive \sqsubseteq \exists Update.Video$$

This policy implies that all *Coder*s and *Manager*s not only can *Update* but they can also *Delete* and *Read* some of the material on *Rui's Semantic Desktop*. As a second example, the following (equivalence version of the) *TAC* rule

$$Knowdive \equiv \forall Video.Update$$

states that all the people in the KnowDive group and therefore all *Coder*s and *Manager*s not only can *Update* but they can also *Delete* and *Read* all *Video*s on *Rui's Semantic Desktop* and *nothing more*.

7 Related Work

The state of the art is definitely *RBAC* [11][10]. The amount of work which has been done on *RBAC* and its level of development is incomparably high compared to that of *RelBAC* (see, e.g., [19, 1, 25] or [5]). Therefore a meaningful comparison can be made only on the basic underlying intuitions. As already hinted in

[10] The *UCON* model [27] deals with temporal and state transition issues which our outside the current scope of *RelBAC*.

the previous sections (and emphasized in the introduction and the conclusions) the main difference between *RelBAC* and *RBAC* is that the former models permissions as ER relations thus making them first class entities (which can evolve independently of subjects and objects), and thus allowing for arity aware access control policies. The further main difference is that *RelBAC* embeds policies directly into a (Description) Logic which allows automated reasoning. These two ingredients are among what we believe the main recipes for addressing the complication of the current new Web based open, highly dynamic applications. From another point of view, we can see *RelBAC* as a *natural* extension of *RBAC* by adding the differences above. Our preliminary experiments make us believe that ultimately, *RelBAC* will be used 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., [4, 18, 24]. Besides the differences in the underlying logic and in in the specifics of the formalizations (an obvious consequence of the differences existing between *RBAC* and *RelBAC*), it is here worth mentioning that in this work the 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 in *RelBAC* to have non-logic experts to handle (logic) policies and to reason about them using state of the art reasoning technologies.

Some researchers have dealt with problems similar in nature to the ones concentrated in *RelBAC*. Thus, for instance, Juri et al. propose an access control solution for sharing semantic data across desktops [10]. They use a three dimensional access control matrix to represent fine-grained policies. The problem is that their solution will not scale since the matrix grows polynomially with objects. Other authors have addressed the problem of access control in open and dynamic environments by adapting *RBAC*. One such approach is [4]. Another approach is the algebra of security policies for access control in [7]. This algebra allows for composing access control policies. S. Agarwal and B. Sprick propose a similar algebra for dealing with the problem of access control with semantic web services [2].

Various papers describe the use of Description Logics in the formalization of access control (see, e.g. [8, 12, 14, 15, 33, 34]). In [34] an early attempt was made by C.Zhao et al. to apply DLs to the representation of policies in the *RBAC*model. In their proposal, *users, roles, sessions* and *permissions* are formalized as DL concepts but *objects* are regarded as encapsulated inside *permissions* together with *operations*. This results in an explosion in the number of permissions and the corresponding difficulty to specify policies about *objects*. Moreover, they proposed to use only the *existential restriction* constructor for *permission assignments*. Another formalization of *RBAC*in DLs was proposed by J.Chae et al. [8], where an *operation* is represented as a DL role. Recently, T.Finin et al. proposed to use OWL[11] as the formalization of *RBAC*in [12]. They provide two ways to formalize a *RBAC*role, as a class or as an attribute. N3Logic

[11] http://www.w3.org/TR/owl-guide/

is used together with DL subsumption reasoning. Authorization decision queries can be answered using DL reasoners in their system.

8 Conclusion

In this paper, we have discussed a new access control model, *RelBAC* and have formalized it using the Description Logic, $\mathcal{ALCQIBO}$. The subject groups, object classes are formalized as concepts and the permissions, which are intuitively the rights to perform certain operations by the subject onto the object, are formalized as Description Logic roles. We also showed the way to define hierarchies inside the subjects, objects and permissions; the way to define general and ground access control rules. *RelBAC* allows for great expressiveness in cardinality control and natural definition in policies. In addition, a popular security property, separation of duties, had been discussed under *RelBAC* in contrast to other access control models. With this paper, we can see *RelBAC* as a novel efficient natural access control model for the Web 3.0.

References

1. Abou El Kalam, A., Baida, R.E., Balbiani, P., Benferhat, S., Cuppens, F., Deswarte, Y., Miège, A., Saurel, C., Trouessin, G.: Organization based access control. In: 4th IEEE International Workshop on Policies for Distributed Systems and Networks (Policy'03) (June 2003)
2. Agarwal, S., Sprick, B.: Access control for semantic web services. icws 0, 770 (2004)
3. Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F. (eds.): The description logic handbook: theory, implementation, and applications. Cambridge University Press, New York, NY, USA (2003)
4. Bertino, E., Catania, B., Ferrari, E., Perlasca, P.: A logical framework for reasoning about access control models. ACM Transactions on Information and System Security 6(1), 71–127 (2003)
5. Bichler, M., Kalagnanam, J., Katircioglu, K., King, A.J., Lawrence, R.D., Lee, H.S., Lin, G.Y., Lu, Y.: Applications of flexible pricing in business-to-business electronic commerce. IBM Systems Journal 41(2), 287–302 (2002)
6. B.Lampson: Protection. In: Proc. 5th Princeton Conf. on Information Sciences and Systems, Princeton, 1971. Reprinted in ACM Operating Systems Rev. 8, 1. pp. 18–24 (1971)
7. Bonatti, P.A., di Vimercati, S.D.C., Samarati, P.: An algebra for composing access control policies. Information and System Security 5(1), 1–35 (2002), `citeseer.ist.psu.edu/bonatti02algebra.html`
8. Chae, J.H., Shiri, N.: Formalization of rbac policy with object class hierarchy. In: Dawson, E., Wong, D.S. (eds.) ISPEC. Lecture Notes in Computer Science, vol. 4464, pp. 162–176. Springer (2007), `http://dblp.uni-trier.de/db/conf/ispec/ispec2007.html\#ChaeS07`
9. Chen, P.P.: The entity-relationship model - toward a unified view of data. ACM Transactions on Database Systems 1(1), 9–36 (1976)

10. Coi, J.L.D., Ioannou, E., Koesling, A., Olmedilla, D.: Access control for sharing semantic data across desktops. In: 1st International Workshop on Privacy Enforcement and Accountability with Semantics (PEAS). Busan, Korea (Nov 2007), 2007/2007_PEAS_desktop_policies.pdf
11. Ferraiolo, D.F., Sandhu, R.S., Gavrila, S.I., Kuhn, D.R., Chandramouli, R.: Proposed NIST standard for role-based access control. Information and System Security 4(3), 224–274 (2001), citeseer.ist.psu.edu/ferraiolo01proposed.html
12. Finin, T., Joshi, A., Kagal, L., Niu, J., Sandhu, R., Winsborough, W., Thuraisingham, B.: Rowlbac: representing role based access control in owl. In: SACMAT '08: Proceedings of the 13th ACM symposium on Access control models and technologies. pp. 73–82. ACM, New York, NY, USA (2008)
13. Giunchiglia, F., Zaihrayeu, I.: Lightweight ontologies. In: Encyclopedia of Database Systems. Springer (2009)
14. Giunchiglia, F., Zhang, R., Crispo, B.: Relbac: Relation based access control. In: SKG '08: Proceedings of the 2008 Fourth International Conference on Semantics, Knowledge and Grid. pp. 3–11. IEEE Computer Society, Washington, DC, USA (2008)
15. Giunchiglia, F., Zhang, R., Crispo, B.: Ontology driven community access control. In: SPOT2009 - Trust and Privacy on the Social and Semantic Web (2009)
16. Haystack: Http://groups.csail.mit.edu/haystack/
17. IRIS: Http://www.openiris.org/
18. Jajodia, S., Samarati, P., Sapino, M.L., Subrahmanian, V.S.: Flexible support for multiple access control policies. Database Systems 26(2), 214–260 (2001), citeseer.ist.psu.edu/jajodia01flexible.html
19. Joshi, J., Bertino, E., Latif, U., Ghafoor, A.: A generalized temporal role-based access control model. IEEE Transactions on Knowledge and Data Engineering 17(1), 4–23 (2005)
20. Li, N., Tripunitara, M.V., Bizri, Z.: On mutually exclusive roles and separation-of-duty. ACM Transactions on Information System Security 10(2), 5 (2007)
21. Li, N., Wang, Q.: 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. pp. 356–369. ACM, New York, NY, USA (2006)
22. Lutz, C., Sattler, U.: The complexity of reasoning with boolean modal logics. In: Wolter, F., Wansing, H., de Rijke, M., Zakharyaschev, M. (eds.) Advances in Modal Logics Volume 3. CSLI Publications, Stanford (2001)
23. Lutz, C., Walther, D.: Pdl with negation of atomic programs. Journal of Applied Non-Classical Logic 15(2), 189–214 (2005)
24. Massacci, F.: Reasoning about security: A logic and a decision method for role-based access control. In: ECSQARU-FAPR. pp. 421–435 (1997), citeseer.ist.psu.edu/massacci97reasoning.html
25. Moyer, M.J., Ahamad, M.: Generalized role-based access control. In: ICDCS. pp. 391–398 (2001)
26. Nepomuk: Http://nepomuk.semanticdesktop.org/
27. Park, J., Sandhu, R.: The ucon¡sub¿abc¡/sub¿ usage control model. ACM Transactions on Information and System Security 7(1), 128–174 (2004), http://dx.doi.org/10.1145/984334.984339
28. Pratt-Hartmann, I.: Complexity of the two-variable fragment with counting quantifiers. J. of Logic, Lang. and Inf. 14(3), 369–395 (2005)
29. Schmidt, R.A., Tishkovsky, D.: Using tableau to decide expressive description logics with role negation. In: Aberer, K., Choi, K.S., Fridman Noy, N., Allemang, D., Lee,

K.I., Nixon, L.J.B., Golbeck, J., Mika, P., Maynard, D., Mizoguchi, R., Schreiber, G., Cudré-Mauroux, P. (eds.) The Semantic Web, 6th International Semantic Web Conference, 2nd Asian Semantic Web Conference, ISWC 2007 + ASWC 2007, Busan, Korea, November 11–15, 2007. Lecture Notes in Computer Science, vol. 4825, pp. 438–451. Springer (2007)

30. Sirin, E., Parsia, B., Grau, B.C., Kalyanpur, A., Katz, Y.: Pellet: A practical owl-dl reasoner. Submitted for publication to Journal of Web Semantics. (2003)
31. Strembeck, M., Neumann, G.: An integrated approach to engineer and enforce context constraints in rbac environments. ACM Transactions on Information and System Security 7(3), 392–427 (2004)
32. Wilikens, M., Feriti, S., Sanna, A., Masera, M.: A context-related authorization and access control method based on rbac:. In: SACMAT '02: Proceedings of the seventh ACM symposium on Access control models and technologies. pp. 117–124. ACM Press, New York, NY, USA (2002)
33. Zhang, R.: RelBAC: Relation Based Access Control. Ph.D. thesis, University of Trento (March 2009)
34. Zhao, C., Heilili, N., Liu, S., Lin, Z.: Representation and reasoning on rbac: A description logic approach. In: Hung, D.V., Wirsing, M. (eds.) ICTAC. Lecture Notes in Computer Science, vol. 3722, pp. 381–393. Springer (2005), `http://dblp.uni-trier.de/db/conf/ictac/ictac2005.html\#ZhaoHLL05`