RELBAC: RELATION BASED ACCESS CONTROL

Fausto Giunchiglia, Rui Zhang and Bruno Crispo

August 2008

Technical Report # DISI-08-040

Also: published in The International Conference on Semantics, Knowledge and Grid (SKG) Bijing, 2008

RelBAC: Relation Based Access Control

Fausto Giunchiglia, Rui Zhang, Bruno Crispo
Dipartimento di Ingegneria e Scienza dell'Informazione, University of Trento
Via Sommarive, 14 I-38050 POVO, Trento Italy
{fausto,zhang,crispo}@disi.unitn.it

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. In this context, metadata may be tags or complex graph structures such as file system or web directories, or (lightweight) ontologies. In turn, users can themselves be tagged by certain properties, and can be organized in complex directory structures, very much in the same way as data. Things are further complicated by the highly unpredictable and autonomous dynamics of data, users, permissions and access control rules. In this paper we propose a new access control model and a logic, 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, e.g., *Role Based Access Control (RBAC)*, is that permissions are modeled as relations between users and data, while access control rules are their instantiations on specific sets of users and objects. As such, access control rules are assigned an arity which allows a fine tuning of which users can access which data, and can evolve independently, according to the desires of the policy manager(s). Furthermore, the formalization of the *RelBAC* model as an Entity-Relationship (ER) model allows for its direct translation into Description Logics (DL). In turn, this allows us to reason, possibly at run time, about access control policies.**

I. 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 [1], Haystack [2], Nepomuk [3]) 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. 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:

- 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 [4], [5]. 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)* [6]). Furthermore a much more refined control on the arity of access control rules must be enforced. In particular, it must be possible to say that, e.g., *m* users in a pre-existing group can access *n* data in a pre-existing 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 mechanisms. 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.

In this paper we propose a new access control model and a logic, called *RelBAC* (for *Relation Based Access Control*) which allows us to deal with this novel scenario. The key idea, which differentiates the *RelBAC* model from the state of the art, is that permissions are modeled as relations between users (called *subjects* in access control terminology) and data (also called *objects*) while access control rules are their instantiations, with arity, on specific sets of users and objects. We define the RelBac model as an *Entity Relationship (ER) model* [7] thus defining permissions as relations between classes of subjects and classes of objects. Finally, and this is the last component of our approach, by exploiting the well known translation of ER diagrams into *Description Logics (DL)* [8] we define a (Description) logic, called the *RelBAC* Logic, which allows us to express and reason about users, objects, permissions, access control rules and policies. In turn, this allows us to reason about policies by using state of the art, off-the-shelf, DL Reasoners, e.g., Pellet [9]. 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

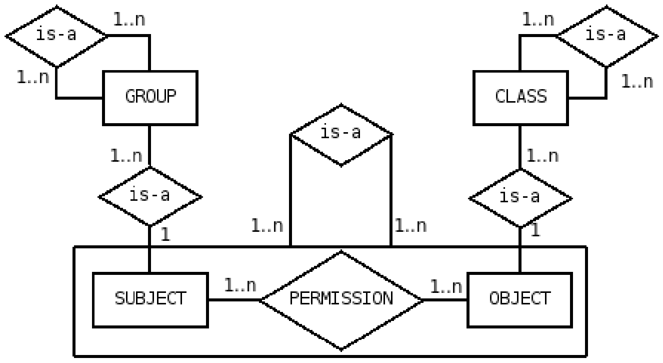

Fig. 1. The ER Diagram of the *RelBAC* Model.

control rule (written in DL) 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 $Student \sqsubseteq \leq 1Use.PC$ states that students can use one and most one PC among all PCs which are available to them. Notice that the above policy states that in different times the same student is allowed to use different PCs.

The rest of the paper is structured as follows. Section II describes the *RelBAC* model. Section III introduces the *RelBAC* logic in all its articulations, and in particular it describes how to define complex sets of users, objects and permissions, and how to define hierarchies, and (general and ground) policies. This section also describes how to represent in *RelBAC* the most common type of policies used so far in access control. Section IV is an early discussion about RelBAC, how to use it in practice, about its underlying ideas and of how it compares with the existing frameworks. Section V describes the related work and Section VI describes the future work and provides some conclusions.

II. The *RelBAC* Model

We represent the *RelBAC* Model with the ER Diagram in Figure 1. Let us analyze its components in detail.

SUBJECT (or USER): a set of subjects, namely the set of those users who can perform operations on objects;

GROUP: any subset, or group, or community of users;

OBJECT: a set of all objects, namely the set of all data (and/ or resources) on which users can perform operations;

CLASS: any subset, or class of objects;

IS-A[1] on SUBJECT, GROUP, OBJECT, CLASS: as from Figure 1, SUBJECT and OBJECT can have many subsets, in a one-to-many relationship while GROUP and CLASS, being in a many-to-many ISA hierarchy with themselves, form a Direct

[1]By IS-A, also written ISA, we mean here any subset of a given set of entities (like in the case of SUBJECT, OBJECT, GROUP, CLASS), or of pairs of entities (as in the case of PERMISSION), thus slightly stretching the original meaning of ISA.

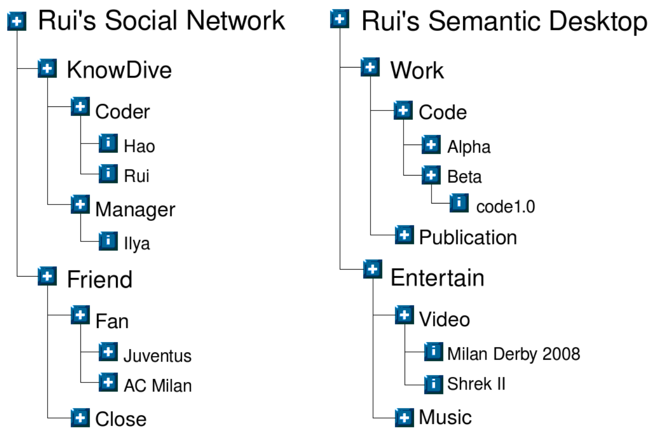

Fig. 2. SUBJECT and OBJECT Hierarchies.

Acyclic Graph (a DAG). This part of the model is what allows us to capture the fact that data and users can be organized in arbitrarily complex hierarchies;

PERMISSION: the intuition is that a PERMISSION is an operation that users can perform on objects. To capture this intuition a PERMISSION is named with the name of the operation it refers to, e.g., *Write*, *Read* operation or some more high level operation, e.g., *CanSign*. According to the ER diagram in Figure 1, a PERMISSION is *a relation* between GROUP and CLASS. This is a first level of flexibility in the definition of access control policies: users and data evolve and, a posteriori, we can add, delete or modify a permission (on the current sets of users and data);

IS-A on the aggregation of SUBJECT, PERMISSION and OBJECT: PERMISSIONS can be hierarchically organized in a many-to-many relation and form a DAG. This captures the intuition that the permission to perform a certain operation may imply the permission of performing another, intuitively weaker, operation. As a simple example, a *Write* permission may imply a *Read* permission, which mathematically means that all the subject/ object pairs in *Write* are also in *Read*, but not necessarily vice versa.

(ACCESS CONTROL) RULE: the intuition is that an ACCESS CONTROL RULE, or simply a RULE, associated to a PERMISSION is the instantiation of PERMISSION to a specific set of users and a specific set of objects. We have two possible types of RULEs. A *user- centric* RULE is defined by saying that a specific set of users is a subset (or a superset or the set) of all subjects which can apply a certain operation to a specific set of objects. Dually, an *object- centric* RULE is defined by saying that a specific set of objects is a subset (or a superset or the set) of all objects which can be applied a certain operation by a specific set of users. Access control rules have an arity. Thus a RULE may be many-to-many. one-to-many, zero-to-many, or of some other arity. In Figure 1 we have represented the case many-to-many just to represent the most powerful relation.

`POLICY`: policies (not represented in the diagram) are sets of access control rules.

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 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 are a subset of those who can have some 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.

The ER Diagram modeling (a part of) the situation in the Figures 2, 3 also providing the missing information is depicted in Figure 4.[2] 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 $Delete$.

III. The *RelBAC* Logic

The possibility to translate ER diagrams into Description Logics allows for a direct formalization of the *RelBAC* model into a family of logics for access control, that we also call

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

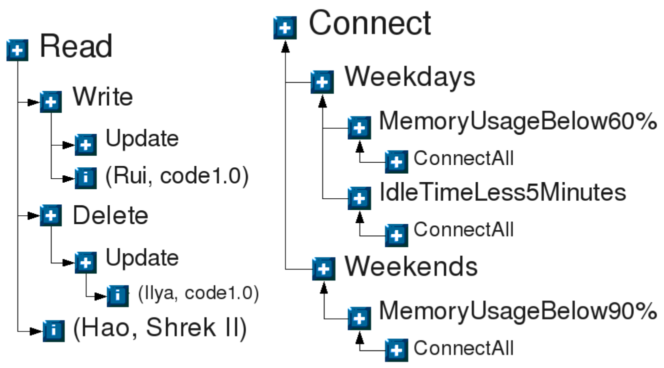

Fig. 3. Permission Hierarchies.

RelBAC. This is a family of logics, rather than a single logic as different policies correspond to different logics, each with its own expressiveness and computational properties (see [8] for a survey on DL).[3] This formalization is quite natural and is done by modeling Subjects (Users) and Objects as Concepts (whose interpretations are, respectively, sets of subjects and objects) and permissions as DL Roles (whose interpretations are binary relations between sets of subjects and sets of objects).

In the following of this section we describe how it is possible to define user groups, object classes and permissions (Section III-A), we show how it is possible to build hierarchies of users, objects and permissions (Section III-B), and then show how to formalize general access control rules (Section III-C) and instance specific access control rules (Section III-D). 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 III-E).

From a logical point of view most of the contents of the following subsections are just a re-statement of the very well known DL language and semantics. The added value is the re-interpretation of DL concepts in access control terms and the evidence that this paper provides of how *natural* this interpretation is.

A. Defining Users, Objects, Permissions

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

$$U_1, ..., U_m \mid \text{(users)}$$
$$O_1, ..., O_n \mid \text{(objects)}$$
$$P_1, ..., P_s \mid \text{(permissions)}$$

where $U_i(i = 1, ..., m)$ are concepts for users, such as $Friend$ or $KnowDive$; $O_j(j = 1, ..., n)$ are concepts for objects,

[3]In this paper we only show the formalization of the *RelBAC* model into the *RelBAC* logic without getting into the details of how different access control policies map into logics of different expressibility and computational complexity. This will be the topic of future papers.

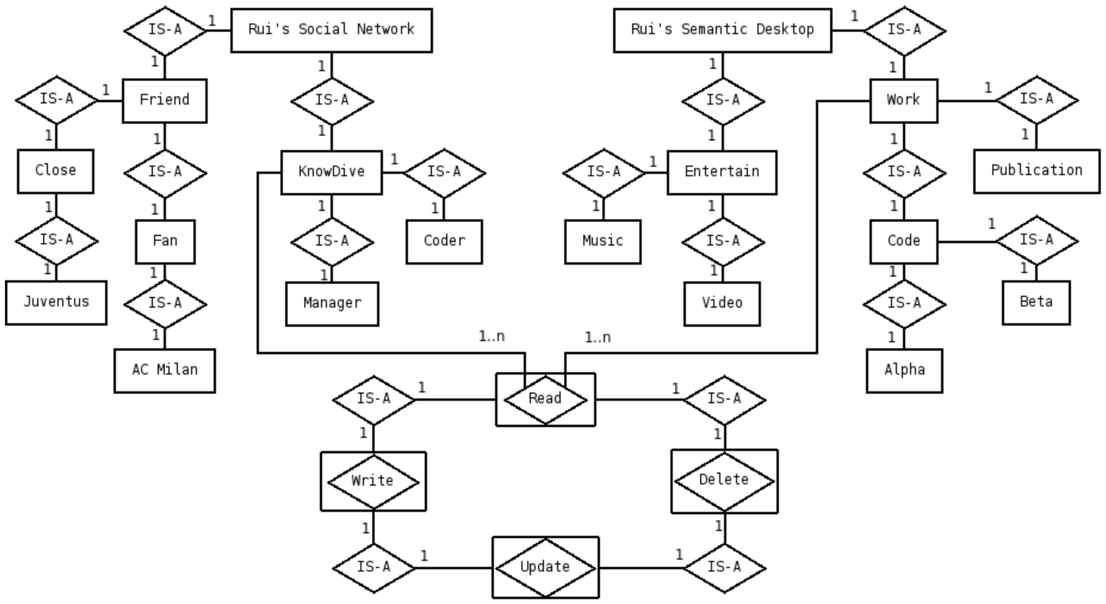

Fig. 4. A portion of the ER Diagram of Figures 2,3.

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 the possibility to define complex groups of users, complex classes of objects and complex relations in terms of the basic sets $User$, $Object$, U_i, O_i and permissions P_i defined above. The *RelBAC* formation rules are:

$$
\begin{aligned}
U_i, O_j \mid & \text{ (atomic concepts for user or object)} \\
P_k \mid & \text{ (atomic permission)} \\
\top \mid & \text{ (universal concept)} \\
\bot \mid & \text{ (empty concept)} \\
\neg U_i, \neg O_j \mid & \text{ (atomic negation)} \\
C \sqcap D \mid & \text{ (conjunction)} \\
\forall P_k.C \mid & \text{ (value restriction)} \\
\forall P_k.\top \mid & \text{ (limited existential quantification)} \\
C \sqcup D \mid & \text{ (disjunction)} \\
\exists P_k.C \mid & \text{ (full existential quantification)} \\
\geq nP_k.C \mid & \text{ (number restriction)} \\
\leq nP_k.C \mid & \text{ (number restriction)} \\
\neg C \mid & \text{ (negation of arbitrary complex concept)} \\
\neg P_k \mid & \text{ (negation of permission)} \\
P_k^{-1} \mid & \text{ (inversion of permission)}
\end{aligned}
$$

A full description of the above constructs is out of the goals of this paper; the reader can refer to any standard book on DL. Let us only make a few general observations. The first is that assertions about users, objects and permissions are embedded in a full propositional language which allows to perform (propositional) reasoning (about policies) (the two distinct items on negation have been kept, even if the second implies the first, as they have very different expressibility and complexity properties). The second is that we have a set of quantificational constructs (i.e., $\forall$, $\exists$ but also $\geq nP$, $\leq nP$) (whose meaning will be made precise below) which allow to express the arity of relations. The third and last is that we can negate permissions where, e.g., $\neg Read$ is the set of all pairs where a user cannot read an object, and that we can also take the inverse of a relation where, e.g., $Read^{-1}$ is the set of all pairs where the object can be read by the user.

The above intuitions can be made precise by using the standard DL semantics as follows. We start with the defintion of *Interpretation* for the language defined above. An *Interpretation* consists of an Interpretation Function $\mathcal{I}$ and a non empty set $\Delta^{\mathcal{I}}$ (the Domain of Interpretation). $\Delta^{\mathcal{I}}$ contains the set of users and objects $User^{\mathcal{I}}$ and $Object^{\mathcal{I}}$. Thus for instance $Hao^{\mathcal{I}}, Rui^{\mathcal{I}} \in User^{\mathcal{I}}$ are the two users of name Hao and Rui while $code1.0^{\mathcal{I}}, Shrek\ II^{\mathcal{I}} \in Object^{\mathcal{I}}$ are examples of data. Then, any user group U_i is interpreted as the set $U_i^{\mathcal{I}} \subseteq User^{\mathcal{I}}$ and similarly, for an object class O_j, we have $O_j^{\mathcal{I}} \subseteq Object^{\mathcal{I}}$. Finally, we interpret a permission P_k as a binary relation $P_k^{\mathcal{I}} \subseteq User^{\mathcal{I}} \times Object^{\mathcal{I}}$. If we consider, for instance, the hierarchy in Figure 3 we have that $\langle Hao^{\mathcal{I}}, Shrek\ II^{\mathcal{I}}\rangle \in Video^{\mathcal{I}} \subseteq Entertain^{\mathcal{I}}$. Notice that in general, we use italic words that begin with lower case letter as instance names, with exceptions.

We can now define the semantics of the above constructs as follows:

$$
\begin{aligned}
\top^{\mathcal{I}} &\supseteq User^{\mathcal{I}} \cup Object^{\mathcal{I}} \\
\bot^{\mathcal{I}} &= \emptyset \\
(\neg U_i)^{\mathcal{I}} &= \Delta^{\mathcal{I}} \setminus U_i^{\mathcal{I}} \\
(\neg O_j)^{\mathcal{I}} &= \Delta^{\mathcal{I}} \setminus O_j^{\mathcal{I}} \\
(C \sqcap D)^{\mathcal{I}} &= C^{\mathcal{I}} \cap D^{\mathcal{I}} \\
(\forall P_k.C)^{\mathcal{I}} &= \{a \in User^{\mathcal{I}} | \ \forall b \ (a,b) \in P_k^{\mathcal{I}} \to b \in C^{\mathcal{I}}\} \\
(\exists P_k.\top)^{\mathcal{I}} &= \{a \in User^{\mathcal{I}} | \ \exists b \ (a,b) \in P_k^{\mathcal{I}}\} \\
(C \sqcup D)^{\mathcal{I}} &= C^{\mathcal{I}} \cup D^{\mathcal{I}} \\
(\exists P_k.C)^{\mathcal{I}} &= \{a \in User^{\mathcal{I}} | \ \exists b \ (a,b) \in P_k^{\mathcal{I}} \wedge b \in C^{\mathcal{I}}\} \\
(\geq nP_k.C)^{\mathcal{I}} &= \{a \in User^{\mathcal{I}} | \ |\{b|(a,b) \in P_k^{\mathcal{I}}, b \in C^{\mathcal{I}}\}| \geqslant n\} \\
(\leq nP_k.C)^{\mathcal{I}} &= \{a \in User^{\mathcal{I}} | \ |\{b|(a,b) \in P_k^{\mathcal{I}}, b \in C^{\mathcal{I}}\}| \leqslant n\} \\
(\neg C)^{\mathcal{I}} &= \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}} \\
(\neg P_k)^{\mathcal{I}} &= User^{\mathcal{I}} \times Object^{\mathcal{I}} \setminus P_k^{\mathcal{I}} \\
(P_k^{-1})^{\mathcal{I}} &= \{(b,a) \in Object^{\mathcal{I}} \times User^{\mathcal{I}} | \ (a,b) \in P_k^{\mathcal{I}}\}
\end{aligned}
$$

where '$|\cdot|$' denotes the cardinality of a set, n ranges over non-negative integers. Let us consider in detail the quantificational constructs. We have that, e.g., $\forall Read.Video$ denotes the set of all users who can read (see) *only* videos, while $\exists Read.\top$ denotes the set of who can read (in Figure 3, something in Rui's Semantic Desktop), $\exists Read.Video$ denotes the set of who can read (see) *at least* a video, $\geq 3Read.Video$ and $\leq 3Read.Video$ denote the set of those who can, respectively, read (see) *at least* or *at most* three videos. The most important observation is that the above constructs not only allow to define complex sets of users out of the basic ones, e.g. $Video \sqcup Music$ but, through the quantifiers, they also allow to define sets with the desired arity. As the following will make clear, this is the basis on which access control rules are defined.

B. 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):

$$A_i \sqsubseteq A_j$$

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

$$
\begin{aligned}
Coder &\sqsubseteq KnowDive \\
Code &\sqsubseteq Work \\
Write &\sqsubseteq Read \\
Connect &\sqsubseteq Weekends
\end{aligned}
$$

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

$$U_1 \geq U_2 \ iff \ U_1 \sqsubseteq U_2$$

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

$$O_1 \geq O_2 \ iff \ O_1 \sqsubseteq O_2$$

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

$$P_1 \geq P_2 \ iff \ P_1 \sqsubseteq P_2$$

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. 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 or objects. Dually, 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 sets of users and objects which are smaller and therefore higher in their partial order are those involved in the more powerful permissions.

C. Defining General Access Control Rules

Access control rules may take one of the following three forms

$$
\begin{aligned}
C &\equiv D \\
C &\sqsubseteq D \\
C &\sqsupseteq D
\end{aligned}
$$

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 III-A. 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:

1) $Friend \sqsubseteq \exists Download.Music$ to state that all close friends can download some music,
2) $Music \sqsubseteq \exists Download^{-1}.Friend$ to state that all music can be downloaded by some friend,
3) $Friend \sqsubseteq \forall Download.Music$ to state that friends can download only music,
4) $Code \sqsubseteq \forall Read^{-1}.KnowDive$ 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:

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

D. 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:

1) $KnowDive(Rui)$ declares Rui to be a member of the group $KnowDive$,
2) $Video(Shrek\ II)$ states that the file $Shrek\ II$ is a video
3) $Download(Hao, Shrek\_II)$ 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:

1) $CloseFriend \sqsubseteq Download : Shrek\ II$ to state that close friends can download $Shrek\ II$,
2) $CloseFriend \sqcap \neg\{Hao\} \sqsubseteq Download : Shrek\ II$ to state that Hao is an exception to the previous policy;
3) $\{Hao, Ilya\} \sqsubseteq Download : Shrek\_II$ to enumerate the two close friends who can download $Shrek\ II$,
4) $(Download : Shrek\_II)(Hao)$ to state that Hao can download $Shrek\ II$,

5) $\exists Update.Beta(Hao)$ to state that Hao can update at least a file of beta code,
6) $\forall Upload.Video(Hao)$ to state that Hao can upload only video,
7) $\geq 10 Update.Beta(hao)$ to state that Hao can update at least ten files of Beta code,
8) $\leq 15 Download.Video(Hao)$ 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:

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

E. 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 has been constructed by Alex Borgida and is defined as follows:

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

We have in fact that

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

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:

1) $Manager \sqsubseteq \forall Code.Update$ states that all project managers can update all the code,
2) $Code \sqsubseteq \forall Manager.Update^{-1}$ 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

$$Manager \equiv \forall Code.Update$$
$$Code \equiv \forall Manager.Update^{-1}$$

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 IV-C).

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,

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

but read Section IV-B on the use of equivalences in the definition of access control rules.

IV. 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.

A. 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 III-E). 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.

B. 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 III-C). 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 palmar. One would like to add the following policy

$$Student \sqsubseteq \exists Use.Palmar$$

but this would lead an inconsistent theory (under the assumption that palmars 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.

C. 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* [10]. 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 admnistrator 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).

D. 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:

1) ground policies.
2) general policies.

Let us consider these two issues in turn. 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 III-C), 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 III-C (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*.

Furthermore, in our first implementation of *RelBAC*, we have implemented user and object directories as Lightweight Ontologies [11] (also called Formal Classifications [12], [13]) and performed some preliminary evaluations of the possibility of automatically classifying users and objects in directories and of automatic generation of permissions based on the user interests, exploiting the Semantic Matching technology [14], [15], [16], [17], [18], [19]. The results, though preliminary, look very promising; future papers will investigate this venue of research. It must be pointed out that this research line, if successful, will allow us to share access control policies using Web standard languages, e.g., OWL.

V. Related Work

The state of the art to which we need to refer to is definitely *RBAC* [6].[4] 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., [21], [22], [23] or [24]). Therefore a meaningful comparison can be made only on the basic underlying intuitions. As already hinted in the previous sections (and emphasized in the introduction and the conclusions) The main difference of *RelBAC* with respect to *RBAC* is that the former models permissions as ER relations thus making them first class objects (which can evolve independently of users and objects), and thus allowing for arity aware access control policies. The further main difference is that this allows for a direct embedding on policies into a (Description) Logic which allows to reason about them. These two ingredients are among what we believe are the main recipes for addressing the complication of the current new GRID and Web open, highly dynamic applications.

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 seem to highlight a substantial similarity in the implementation (of ground policies) and in the user interaction (see Section IV-D), 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., [25], [26], [27]. Besides the differences in the underlying logic and in in the specifics of the formalizations, in part also an obvious consequence of the differences existing between the *RBAC* and the *RelBAC* models, it is here worth mentioning that in this previous 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 (the SAT technology - used within DL reasoners - is by far the most advanced technology and the one mostly used in real world applications).

Some work has also been done in formalizing *RBAC* in DL. Thus, for instance, DL is used in [28] in order to formalize relations as binary roles while, more recently, Jung-Hwa Chae et.al use DL to formalize the object hierarchy of *RBAC* [29]. This work is again very different from ours as here DL is just

[4]The *UCON* model [20] deals with temporal and state transition issues which our outside the current scope of *RelBAC*.

another logic used to reason about *RBAC* instead of *the* logic designed to express (*RelBAC*) policies.

Other researchers have dealt with the problem we are interested in. Thus for instance, Juri et al. propose an access control solution for sharing semantic data across desktops [30]. 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 [25]. Another approach is the algebra of security policies for access control in [31]. 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 [32].

VI. Conclusions and Future work

In this paper, we have proposed *RelBAC* a model and a logic for access control designed for open, highly dynamic environments, where users and data are organized in complex graphs (DAGs or more simply directories) and where permissions are handled as first class objects, largely independently from users and objects.

The main novelty relies on the fact the *RelBAC* access control model is designed as an ER model and that permissions are modeled as ER relations. This allows, among other things, to express policies which take into account the arity of permissions, and for a direct translation of the *RelBAC* model into a Description Logic, that we call the *RelBAC* Logic. This, in turn, opens up various possibilities that we can be summarized as follows:

1) It should be possible to change policies at run-time, while the system is in operation and it should be possible to do this by using a simple interface where the administrator would see directories and relations between directories. Even if our policy language is logic based, the administrator will not need to know any logic;
2) it should be possible to reason about policies at run-time, after any policy change, thus being able to adapt to the continuous system dynamics;
3) It should be possible to implement the policy reasoning system by largely re-using the large amount of work on the expressibility, computational complexity and general purpose off-the- shelf DL reasoners. Whether DL reasoners will be directly usable or whether we will need to develop a dedicated reasoner is still an open issue. The preliminary evaluations on using off-the-shelf DL reasoning systems do not look encouraging.

As part of the immediate future work, we plan to investigate the following directions:

1) modeling of the most common and important properties (e.g., static or dynamic separation of duties), definition of the DL logic needed to represent them and the consequent definition of their computational complexity. Notice that we should be able to do it uniformly differently from what is currently the case in access control (see, e.g., [33], [34]);
2) further development and consolidation of our *RelBAC* implementation and
3) its application in some pilot applications. At the moment we have started using it inside a desktop application developed by our group;
4) experimentation with the state of the art DL reasoners in order to evaluate their direct re-usability in this application scenario and, to conclude,
5) development of some automatic techniques for automatic policy generation and maintenance based on our formal classification and semantic matching technology cited in Section IV-D.

Acknowledgements

This work has been possible only because of the great environment and great people working in the Knowledge Diversity group (http://dit.unitn.it/ knowdive/). The interaction with these people has provided food for our thoughts: Fabio Massacci, Viktor Pravdin, Mikalai Yatskevich, Pavel Shvaiko and Ilya Zaihrayeu.

Alex Borgida is strongly thanked for his help on the fromalization of the *TAC* rule.

References

[1] IRIS, "http://www.openiris.org/."
[2] Haystack, "http://www.openiris.org/."
[3] Nepomuk, "http://nepomuk.semanticdesktop.org/."
[4] M. Wilikens, S. Feriti, A. Sanna, and M. Masera, "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*. New York, NY, USA: ACM Press, 2002, pp. 117–124.
[5] M. Strembeck and G. Neumann, "An integrated approach to engineer and enforce context constraints in rbac environments," *ACM Trans. Inf. Syst. Secur.*, vol. 7, no. 3, pp. 392–427, 2004.
[6] 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*, vol. 4, no. 3, pp. 224–274, 2001. [Online]. Available: citeseer.ist.psu.edu/ferraiolo01proposed.html
[7] P. P. Chen, "The entity-relationship model - toward a unified view of data," *ACM Trans. Database Syst.*, vol. 1, no. 1, pp. 9–36, 1976.
[8] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, Eds., *The description logic handbook: theory, implementation, and applications*. New York, NY, USA: Cambridge University Press, 2003.
[9] 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.
[10] B.Lampson, "Protection," in *Proc. 5th Princeton Conf. on Information Sciences and Systems, Princeton, 1971. Reprinted in ACM Operating Systems Rev. 8, 1*, 1971, pp. 18–24.
[11] G. F. and Z. I., "Lightweight ontologies," in *Encyclopedia of Database Systems*, S. LNCS, Ed., 2008.
[12] Z. I. Giunchiglia F., Marchese M., "Towards a theory of formal classification," in *Workshop on Contexts and Ontologies: Theory, Practice and Applications (CandO 2005), 20th National Conference on Artificial Intelligence (AAAI-05)*, Pittsburgh, Pennsylvania, USA, July 9-13 2005 2005.
[13] ——, "Encoding classifications into lightweight ontologies," *Journal of Data Semantics*, vol. 8, 2007.

[14] G. F., Y. M., and S. P., "Semantic matching: Algorithms and implementation," *Journal on Data Semantics*, pp. 1–38, 2007.

[15] G. E. Giunchiglia F., Yatskevich M., "Efficient semantic matching," in *Proceedings of the 2nd European semantic web conference (ESWC'05)*. LNCS, Springer Verlag, 2005.

[16] Y. M. Giunchiglia F., "Element level semantic matching," in *Meaning Coordination and Negotiation workshop at ISWC'04*. Hiroshima, Japan, November 2004.

[17] F. Giunchiglia, F. McNeill, M. Yatskevich, J. Pane, P. Besana, and P. Shvaiko, "Approximate structure preserving semantic matching," in *Proceedings of the 7th International Conference on Ontologies, DataBases, and Applications of Semantics (ODBASE 2008)*. LNCS, Springer Verlag, 2008.

[18] G. F., S. P., and Y. M., "Semantic schema matching," in *Proc. OTM Confederated International Conferences, CoopIS*, V. . Springer, LNCS, Ed., 2005, pp. 347–365.

[19] Y. M. Giunchiglia F., Shvaiko P., "Discovering missing background knowledge in onology matching," in *17th European Conference on Artificial Intelligance - ECAI 2006*, L. Springer, Ed., 2006.

[20] J. Park and R. Sandhu, "The UCON$_{ABC}$ usage control model," *ACM Trans. Inf. Syst. Secur.*, vol. 7, no. 1, pp. 128–174, 2004.

[21] J. Joshi, E. Bertino, U. Latif, and A. Ghafoor, "A generalized temporal role-based access control model." *IEEE Trans. Knowl. Data Eng.*, vol. 17, no. 1, pp. 4–23, 2005.

[22] 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.

[23] M. J. Moyer and M. Ahamad, "Generalized role-based access control," in *ICDCS*, 2001, pp. 391–398.

[24] 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*, vol. 41, no. 2, pp. 287–302, 2002.

[25] E. Bertino, B. Catania, E. Ferrari, and P. Perlasca, "A logical framework for reasoning about access control models," *ACM Trans. Inf. Syst. Secur.*, vol. 6, no. 1, pp. 71–127, 2003.

[26] S. Jajodia, P. Samarati, M. L. Sapino, and V. S. Subrahmanian, "Flexible support for multiple access control policies," *Database Systems*, vol. 26, no. 2, pp. 214–260, 2001. [Online]. Available: citeseer.ist.psu.edu/jajodia01flexible.html

[27] F. Massacci, "Reasoning about security: A logic and a decision method for role-based access control," in *ECSQARU-FAPR*, 1997, pp. 421–435. [Online]. Available: citesader.ist.psu.edu/massacci97reasoning.html

[28] C. Zhao, N. Heilili, S. Liu, and Z. Lin, "Representation and reasoning on rbac: A description logic approach." in *ICTAC*, ser. Lecture Notes in Computer Science, D. V. Hung and M. Wirsing, Eds., vol. 3722. Springer, 2005, pp. 381–393.

[29] J.-H. Chae and N. Shiri, "Formalization of rbac policy with object class hierarchy." in *ISPEC*, ser. Lecture Notes in Computer Science, E. Dawson and D. S. Wong, Eds., vol. 4464. Springer, 2007, pp. 162–176. [Online]. Available: http://dblp.uni-trier.de/db/conf/ispec/ispec2007.html# ChaeS07

[30] 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.

[31] P. A. Bonatti, S. D. C. di Vimercati, and P. Samarati, "An algebra for composing access control policies," *Information and System Security*, vol. 5, no. 1, pp. 1–35, 2002. [Online]. Available: citeseer.ist.psu.edu/bonatti02algebra.html

[32] S. Agarwal and B. Sprick, "Access control for semantic web services," *icws*, vol. 0, p. 770, 2004.

[33] N. Li and M. V. Tripunitara, "Security analysis in role-based access control," *ACM Trans. Inf. Syst. Secur.*, vol. 9, no. 4, pp. 391–420, 2006.

[34] N. Li, M. V. Tripunitara, and Z. Bizri, "On mutually exclusive roles and separation-of-duty," *ACM Trans. Inf. Syst. Secur.*, vol. 10, no. 2, p. 5, 2007.