PhD Dissertation

International Doctorate School in Information and Communication Technologies

DIT - University of Trento

RelBAC-Relation-Based Access Control

Rui Zhang

Advisor:

Prof. Fausto Giunchiglia

Università degli Studi di Trento

February 2009

Abstract

The Web 2.0 is making everything on the web more interactive, more social and more dynamic. Nowadays, unknown users are from dynamic online communities; new resources emerge in different granularity and scales; even relatively static permissions are evolving with new contexts; everything has changed due to the new information technologies. However, the privacy issue is inevitable: How to control the access in the new web scenarios?

Facing these challenges, we propose a new model RelBAC (Relation-Based Access Control), for the access control problem of the web. The main feature of RelBAC is that permissions are modeled as binary relations between the users and the resources. An Entity Relationship (ER) model is used to illustrate RelBAC and by exploiting the formalization of ER models in Description Logics (DL), the model is formalized with a domain specific DL with access control policies encoded as DL formulas. In contrast to other access control model formalizations, RelBAC formalizes permissions as DL roles rather than unary concepts. The logical representation of RelBAC brings automated reasoning to facilitate design and management. Lightweight Ontology theory is exploited to encode the formerly heterogeneous knowledge of users, resources and permissions into ontologies with unified hierarchical structures. Partial orders are defined to formalize these hierarchies and result in free inheritance of user and resource memberships and also free permission propagation. Semantic Matching techniques are used to find the semantic similarities between these lightweight ontologies which in turn facilitate the design and reuse of access control policies.

Keywords
Relation-Based Access Control, Description Logics, Semantic Web, Lightweight Ontologies, Semantic Matching

Contributions and Publications

The work has been developed in collaboration with many people and in particular with: Fausto Giunchiglia, Bruno Crispo and Ilya Zaihrayeu.

The contributions of this thesis are as follows:

- A novel access control model *RelBAC* (short for Relation-Based Access Control) is proposed for the access control problem in dynamic web scenarios. *RelBAC* models permissions as binary relations and hierarchies with partial orders. The ER model of *RelBAC* makes it easy to be integrated into system design.

- *RelBAC* is formalized with access control domain specific Description Logics (DL) in permissions formalized as DL roles, partial orders formalized with subsumption axioms, and access control rules formalized into DL formulas. Automated reasoning about access control policies can be performed on *RelBAC* through off-the-shelf DL reasoners. Some preliminary evaluations are made upon automated generated access control domain ontologies.

- Extendability of *RelBAC* is shown by representing other access control models such as RBAC (short for Role-Based Access Control) and PBAC (short for Purpose-Based Access Control).

- The theory and algorithms of Lightweight Ontologies are applied on *RelBAC* to represent the structured knowledge about users, resources and permissions. The heterogeneous structured knowledge bases are formalized into lightweight ontologies to provide free paths for permission propagation and membership inheritance.

- We use Semantic Matching techniques among the knowledge of users, resources and permissions to find the semantic similarities to guide the design and reuse of access control policies.

- In our first implementation of *RelBAC*, we have encoded user and resource directories as lightweight ontologies and performed some preliminary evaluations on reasoning about these ontologies.

Part of the materials of this thesis has been published as follows.

- [44] Fausto Giunchiglia, Rui Zhang, and Bruno Crispo. Relbac: Relation based access control. In *SKG'08: Proceedings of the 2008 Fourth International Conference on Semantics, Knowledge and Grid*, pages 311, Washington, DC, USA, 2008. IEEE Computer Society.
- [45] Fausto Giunchiglia, Rui Zhang, and Bruno Crispo. Ontology driven community access control. Submitted to *SPOT2009 - Trust and Privacy on the Social and Semantic Web*, to appear 2009.
- [88] Rui Zhang, Bruno Crispo, and Fausto Giunchiglia. Relbac:design and run time reasoning about web access control policies. Submitted to *IEEE International Symposium on Policies for Distributed Systems and Networks* (POLICY09), to appear 2009.
- [87] Rui Zhang. Automatic Access Control Rule Generation via Semantic Matching. Accepted as poster in *Workshop on Matching and Meaning 2009*, to appear 2009.

Acknowledgments

I would like to express my earnest gratitude to my adviser, Fausto Giunchiglia. It is he that led me to the entrance to scientific research of Artificial Intelligence. His insightful suggestions, generous encouragements and considerate instructions accompanied me along the way of my PhD study. Through countless hours and endless patience, he taught me how to do research, from the very start as to find a problem, shape the idea, to a higher stage as to write an article, present a paper, and finally build a solution and accomplish this thesis. He is an experienced scientific supervisor, and a life guider as well. His advices always penetrate the intracity like the sunshine through the fogs. His optimism, patience and precision have influenced and will always benefit me all my life.

I am grateful to all the external thesis committee members: Alessandro Artale and Massimo Benerecetti for the time and energy spent on reviewing the thesis and the precious detailed suggestions.

I would thank Bruno Crispo who helped me a lot to understand the field of Access Control. Talks and reviews greatly shortened the warming-up to know a brand new area.

I would appreciate to my friends, colleagues and many other people as this thesis consists of their contributions through comments, discussions, encouragements, advices and all other ways.

Last but not least, I would like to show my everlasting thanks to my parents for understanding my years away from their sides and for their endless love to support me without any complain. I should specially acknowledge my wife, Gao Wenjuan who sacrificed her career and accompanied me through all this period of pursuing my PhD. Her considerate inspirations and blind faith have been the infinite power of my step forwards.

Contents

List of Tables

List of Figures

Introduction

Problem

Access control is a specific problem in the security domains. In the 'Information Age', it usually refers to control the access to digital web resources. Nowadays, access control evolves from the original physical property control into a big problem covering three aspects: *authentication, authorization* and *audit*, which correspond respectively to the questions such as 'whether the user is the one that she claims', 'whether she is allowed to access something' and 'how she is accessing or has accessed the resource'. In this thesis, we focus on the aspect of *authorization* of access control.

In the new web scenarios, access control is far more complex than it used to be. Resources grow rapidly in size and types (e.g., blogs, web directories, tags, semantics, etc.). Most users are from the Internet that one has never met face to face (e.g., online communities, social networks, etc.). What makes it more interesting is that the access right itself becomes complicated with various of attributes and contexts (e.g., time, location, purpose, etc.). All these knowledge are no longer static and centralized because of the new technologies (e.g., Web 2.0, peer-to-peer, eCommerce, etc.). People are no longer only passive information receivers but contributors to the web via FaceBook[27], Flickr[31], Wiki[83], Blogger[11], eBay[24], etc.

As a branch of computer security, access control has been studied for many years, and many access control models have been proposed. However, existing models are not enough to capture the dynamics in the new web scenarios.

Solution

In order to capture the dynamic scenarios of the Information Age, we propose a new access control model, *RelBAC* (short for Relation-Based Access Control). It is a compact, flexible, natural and formalized model in order to fit the needs to control access in the dynamic web environments.

RelBAC is illustrated in a compact Entity Relationship (ER) [19] diagram and formalized in Description Logics (DL) [4]. Therefore, it has flexible extensibility for attributes and contexts. An important feature is that *RelBAC* represents permissions naturally as binary relations and formalizes them as DL roles. Yet another feature is that partial orders are defined to model the generalization relations between subject groups (applicable also between resource classes and between permissions), and formalized as DL subsumption axioms. The logical formalization of *RelBAC* makes it possible to transform access control problems into reasoning tasks for off-the-shelf reasoners. In *RelBAC*, access control tasks such as hierarchy management, security property enforcement and access control decision are no longer error-prone with the help of automated reasoning.

The theory and algorithms of Lightweight Ontologies [**?**] are applied on *RelBAC* to represent the structured knowledge about users, resources and permissions. Encoded as lightweight ontologies, these knowledge structures offer *free* membership inheritance and permission propagation (no extra policy necessary for inheritance or propagation). We use Semantic Matching techniques [41] to find semantic similarities among the knowledge of users, resources and permissions and propose new ways to use these semantic similarities in design and reuse of access control policies.

Thesis Structure

This thesis consists of the following parts.

The new challenges of the Information Age for access control are studied in Chapter 1. To show the great need for a new access control model, a small survey is provided on the state of the art access control mechanism of social network applications.

Chapter 2 analyzes a set of existing access control models such as AM, ACL, RBAC, etc. RBAC, as a prevailing state of the art access control model in many real world applications, is carefully studied. Its advantages such as permission inheritance in role hierarchy inspire our modeling.

In Chapter 3, *RelBAC* is introduced in an Entity-Relationship diagram. Together, a Description Logic based, access control domain specific logical framework is used to formalize the model.

Automated reasoning with *RelBAC* at design and run time is discussed in Chapter 4. We show in this chapter how *RelBAC* facilitates administration by transforming access control tasks such as membership and hierarchy management, security property enforcement, access control decision etc. into automated reasoning tasks for the reasoner.

RelBAC is adaptive to capture existing access control models. Chapter 5 shows how to capture the RBAC model with small extension of *RelBAC*. Further more, by capturing the PBAC model, *RelBAC* shows its powerful adaptiveness and extensibility to attributes and contexts.

In Chapter 6, a semantic web technology, i.e. the theory of Lightweight Ontologies, is applied to build tree-like structures out of the Internet users, web resources and contextual permissions. Control knowledge and provide free paths for permission propagation and membership inheritance.

Semantic Matching is used to help design and reuse access control rules in Chapter 7 with the semantic similarities found in the matching.

Evaluation results are shown in Chapter 8. Demo ontologies of access control domain are designed and generated for the test. The preliminary results bring us confidence and challenges for future work.

The structure of a *RelBAC* based system is also provided from an implementation point of view in Chapter 9. Some practical problems such as closed world assumption [73] and the implementation of the 'All' rules are also discussed here.

Chapter 10 concludes the thesis and highlights the future work.

As this thesis might be interesting for researchers from security and engineering domains, a general introduction to the basic Description Logic syntax and semantics is attached in Appendix A in case of necessity.

Chapter 1

New Challenges for Access Control

The idea of access control dates back to hundreds of years ago in order to protect private property. The owner controls the access of others according to her own will. Nowadays, information explodes that various kinds of data appear and vanish on the Internet. Internet users are hidden at the back of their screens, and the question whether to permit or deny their requests cannot be answered through face to face experiences. What makes things more complicated is that the way of access is not only read or update, but rather contexts related such as 'compute when system load low', 'comment to a tagged blog', etc. These new changes make access control a more interesting and important topic in the computer security domain.

In this chapter, we first introduce the formal definition of an access control problem in Section 1.1, and list the new challenges for access control brought by the new information technologies in Section 1.2. Section 1.3 surveys the access control mechanisms of the current online social network communities. Section 1.4 gives a short summary and lists the challenges and lessons for our model.

1.1 The Access Control Problem

Intuitively, access control is to permit or deny the access of a particular resource by a particular entity. Usually, the entity applied for the access is called a *subject*; the resource is called an *object* and the way to access is called an *operation*. In the field of computer security nowadays, a *subject* could be an Internet user, an online community, a terminal or even a thread of a running program. Information Age is a time of information explosion. Therefore, an *object* can be any kind of data such as web pages, blog entries, web directories, computational capabilities and even meta-data such as tags, logs, indexes, semantics, etc. According to the changes of the subject and the object, an *operation* is no longer simply 'access' only, but varies corresponding to the field of application such as in file systems, it could be *read, update, execute, etc.*; while in ubiquitous computing, it could be *compute, delegate, etc.*

In today's computer security, an access control system usually consists of three essential services as *authentication*, *authorization* and *audit*. *Authentication* verifies whether a user is whom she claims to be; *authorization* assigns and checks the permission to a user that she can (not) perform some operation on some resource (which is the original intention of access control); *audit* manages the security logs on what a user is doing and what she has done. In this thesis we only focus on *authorization* service.

The formal definition of an access control problem is as follows:

Definition 1. Given three sets: *subject S, object O* and *operation P*, access control means to assign a boolean value $\{T, F\}$ to each triple $(s, p, o) \in S \times P \times O$. A 'T' value indicates permission of subject s to perform p on object o, while an 'F' value indicates prohibition.

The assignment $(s, p, o) \to \{T, F\}$ could take the form of (s, p, o, T) or (s, p, o, F) as *permit/deny policy* stored in an access control system. For

Algorithm 1: Access Control Decision

```
Data: s: Subject; o: Object; p: Permission; f: boolean; KB: set of wff;
Input: Subject s, Operation p, Object o
Output: boolean
1 if (s,p,o,F) exists in KB then
2     return false;
3 else
4     if (s,p,o,T) exists in KB then
5         return true;
6     else
7         return false;
8     end
9 end
```

security reasons, deny policy is usually supposed to be prior to a permit policy with the same (s, p, o) triple. An authorization algorithm could be as Algorithm 1. The return value 'False' on Line 7 indicates that for a given request triple (s, p, o), if neither permit nor deny policy is found, the system should deny the access.

1.2 New Challenges

New web technologies bring new challenges to access control. In this section, we show these challenges with different scenarios of web resources, ubiquitous computing and online social networks from the perspective of resource, environment and user.

1.2.1 Web Resource

Information explosion produces vast number of various web resources. Nowadays, huge amount of information in numerous types are available to everyone connected to the Internet without any effective access control.

Violence and sex are exposed to teenagers; malicious peers are pretending to be resource providers; private photos and movies are spread to the public; so on and so force. It is high time we have a control mechanism on top of these resources.

However, these resources turn out some challenges that classical access control models cannot overcome.

The size of web resources varies greatly from tens of entries on a personal blog to a professional web directory of millions of URLs. For example, web directory is a special kind of website that provides a list of websites organized by category and subcategory such as Yahoo! [85] and Dmoz [22]. The original Dmoz divisions were 'Adult', 'Arts', 'Business', ..., altogether 15 categories. Subcategories under these divisions undergo a gradual evolution. Dmoz grows to 4 million URLs by Dec 3rd, 2003. Therefore, a model adaptable to both small and large resources is in great need.

The types of web resources increase together with its size. Each new format has its own features and attributes that should be considered by the access control mechanism. For example, online media is gradually popular nowadays. Musics and videos are published online with or without official authorization. Links are redirected from site to site and it is hard to find the source publisher and restrict the 'free-riding'. Another example is the commenting service provided by Web 2.0. Blogs and personal achievements (photos, videos, etc.) can be commented. Rather than leave all the comments visible for all visitors, these comments themselves are data on which access should be controlled such as to enforce a policy that '*a registered user can comment on one topic only 3 times a day*'. Yet another important type is the meta-data such as tags and semantics. For example, tags such as 'My favorite' or 'Cartoon' may disclose privacy issues and should be considered an important type of resource.

In short, various data in different granularity and scales ask for a flexible access control model.

1.2.2 Ubiquitous Computing

Ubiquitous computing [48] has been proposed as a post-desktop model of human-computer interaction in which information processing has been thoroughly integrated into everyday objects and activities.

All models of ubiquitous computing share a vision of small, inexpensive, robust networked processing devices, distributed at all scales throughout everyday life. For example, an intelligent meeting room scenario consists of sensors for environment such as time, temperature to adjust the temperature of the meeting room when there is actually a meeting going on. A human face recognition device is used to recognize the meeting members from the registration profiles. A Mac[1] address recognition mechanism is used to identify the computer of a user to her registration and so forth. Small devices should share information and cooperate in an integrated way.

The scenario proposes the following challenges as a changing environment:

Time The exact meeting time may anticipate or delay from the common schedule and the face recognition device should interact with the time monitor in order to adjust the information of 'meeting time'.

Position The physical position of a user can be recognized through the face recognition device and serves as a parameter to verify the user's access request. The digital devices such as laptop or PDA that can initiate an access request in respect of the user can be coincidently recognized in the same meeting room (e.g., with registered Mac). It is a precondition that the device can delegate the user while being used by the owner.

[1]IEEE 802 standard.

System The system load such as memory usage, connection number, data flows, etc. is evolving environment parameters for access control decision. With the hardware development, these parameters are monitored and returned from different devices.

Thus the changing environment asks for adjustable access control model to take use of these environment parameters.

1.2.3 Social Network

With the help of Internet, people get to know more and more others without traditional face to face contact. There exist now many large social network portals such as Blogger [11], FaceBook [27], LinkedIn [60], MySpace [66], Flickr [31], etc. that offer new ways to make friends. The spirit of Internet, 'share', calls up all the people to participate the online social networks. People with common interests or opinions get together. These 'known' friends, in addition to many other 'unknown' friends, are potential requesters that an access control system should concern.

The following features of the 'new' social networks complicate the access control problem in the perspective of the subject.

Unknown Almost all the friends on the current online social networks have never met each other before. Although sharing the same interests, they are just icons twinkling on an instant messenger window or names under a comment of certain blog, photo or video.

Heterogeneous One's social networks may be different from site to site. The 'dating sites[2]' do not have a common standard for the social networks. Moreover, different social network portals focus on different topics such as Blogger offers blog service while Flickr deals with

[2]nick for those web sites where people can make new friends

photos. Therefore, even for the same person, her social information structures are heterogeneous on different sites and almost impossible to import or combine.

Evolving People are alive and incline to move optionally in contrast to relatively static data. The structure of the social networks may evolve with new groups emerging and obsolete ones vanishing. Besides, social relations become invalid with time passing by.

The user structures are more dynamic due to these aspects and require an expressive model adaptive to this feature.

1.3 Current Applications

In this section, we show the results of a survey on the access control services provided by popular social network portals.

The '$\surd$' in Table 1.1 shows that the online social network in column offers the access control service described in the corresponding row. Here, 'Profile' control consists of the options on whether to show the online 'Status', user 'Photo', and 'Basic Info' such as sex, birth, etc. 'Multimedia' control offers resource management such as traditional online media (e.g., 'Photo' and 'Video') in addition to the new Web 2.0 data (e.g., 'Post' and 'Comment'). 'Structure' control refers to the service to organize resources such as photos into sets in order to control the access on sets rather than single items. 'Search' control allows users to choose whether their data would be visible in others' searching. 'User Control' consists of 'Age' control which validate the age of the content requesters, and 'Group' is a further mechanism provided in Bebo [7] and Friendster [33] that potential users can be collected in to a group so as access right can be assigned once for all group members. Most online social network applications aim to invite new users to join the network with a method we call 'Email Explore'

Table 1.1: Access Control of Current Social Network Portals

	Facebook	**MySpace**	**LinkedIn**	**Flickr**	**Bebo**	**Friendster**
Profile	√	√	√		√	√
Status	√	√	√		√	√
Photo	√		√			√
Basic Info	√	√				√
Multimedia	√			√	√	
Photo	√			√	√	
Video	√				√	
Post	√				√	
Comment	√			√	√	
Structure				√		
Search	√	√	√			
User Control		√			√	√
Age		√				
Group					√	√
Friend Control	√	√	√	√	√	√
Email Explore	√	√	√	√	√	√
Disturbance						√

i.e. to send invitation mail to all the contacts of a registered email address. This mechanism sends email automatically to all addresses on the contact list, such as a notification letter to those already registered email address or an invitation letter to a non-registered address.

From the survey, we see that:

- There is no unified standard for access control services of online social networks. The social network portals just offer some access control mechanisms that they think the user would need.

- Most social network portals in our survey provide the 'Email Explore' service in order to use the contact list of a registered email to invite more friends to this specific online social network. This is a convenient way to invite contacts to join the social network, but might disturb those not interested. Few of the portals (e.g., only Bebo and Friendster) in our survey provide disturbance control by which one has options to accept whose notification message/email or whether to accept it.

- Most social network applications in our survey focus on profile privacy. Some (e.g., Facebook, Friendster) even provide fine-grained control on profile entries such as personal photo, online status, status update or other basic information.

- Social networks with different interests provide different control mechanisms. For example, Flickr provides control on grouped photos while MySpace neglects everything about photo because Flickr is known as a photo sharing site while MySpace focuses more on friends making.

- Some portals offer control based on requesters' age, but no mechanism to verify whether the user information filled during registration is genuine. Therefore, it may not be as helpful as expected.

- Few support grouping of users. For example, Friendster groups users according to geographical information (one group for a continent). However, it is predefined and unchangeable, so it is not able to model the evolving of users in the social networks.

- Few support structured resource management and control. Only Flickr offers arbitrary virtual photo groups and allows to apply predefined control policies on the group members once for all. But it is quite simple that no further structure is supported such as the tree-like file classification in a personal computer (PC) operating system.

- Most of the access control interfaces are complex and not user friendly, some are very hard to find.

Thus, a future access control system for the web should offer the following features:

- Natural and user friendly interface. For example, people are familiar with the folder trees of PC operating systems and we should offer control on top of these tree-like structures.
- Flexible management. If we can offer a user friendly interface for data, why cannot we do it on friends? To the best of our knowledge, there are no social network portals that offer friend management with a flexible manner that captures the evolving feature of the social networks.
- Extensible structure modeling. No matter how we manage to predefine the structure for data and friend management, the web is evolving and may need more in the future. Therefore, we should support an extensible model for these structure dynamics.

1.4 Summary

With the maturity of Web 2.0, information technologies are no longer theoretical terminologies, but real world applications. They bring not only more friendly web interactions, but also new challenges for access control. In this chapter, we studied these challenges from different perspectives such as object, environment and user; and made a survey on existing access control mechanisms on popular social network portals.

We will look through existing access control models in Chapter 2 to justify that they are not suitable for the dynamic web scenarios. Therefore, we introduce our model, *RelBAC* (in Chapter 3).

Chapter 2

Access Control Models

The meaning of access control has changed over years. Initially, access control usually means to restrict the physical access to a facility, place or spot to authorized persons. This is enforced mainly by a physical security guard. Then, with the development of electronic devices, access control has evolved into the usage of physical card access systems such as IC (Integrated Circuit) card or even biometric devices. Nowadays, in computer security, access control further evolves into the composition of three parts: authentication, authorization and audit. Access control authentication devices evolve to include ID and password, digital certificates, security tokens, smart cards and currently biometrics. Access control audit concerns the access procedure and results with a log for future analysis. In this thesis we only focus on the authorization of access control which means to assign and verify whether someone has the right to access something.

Access control models evolve with its meaning. At the very beginning, there is no model but only tuples as discussed in Section 1.1. Then, *access control matrix* appears in [10]. Afterwards, *access control list* emerges to specify the user identities and the privileges granted to them. Later it evolves to *access right digits* attached to the resources in an operating system such as Linux. Then, RBAC [74] model is invented for large enterprise solutions. Furthermore, RBAC has been extended into different models such as PBAC[15], GRABC[65], TRBAC[8], GTRBAC[55], etc. Recently, usage control model (Ucon) is proposed in [72] to formalize obligations, conditions, continuity (ongoing controls) and mutability in addition to authorization.

In this chapter, we make a survey on the existing access control models. The early access control models are surveyed in Section 2.1 including access matrix (AM), access control list (ACL), discretionary access control (DAC), mandatory access control (MAC) etc. Role-based access control (RBAC), as the prevailing important access control model is discussed in Section 2.2. Then we summarize the state of the art formal representations of access control models in Section 2.4 and collect the inspiring ideas of these models and formalizations in 2.5.

2.1 Early Access Control Models

Ever since its emergence long long ago for the first surplus property, continuous efforts have been devoted to improving access control. From the user ability point of view, access control models could be divided into discretionary or mandatory access control. From the core component point of view, there exist AM, ACL, etc. In general, early access control models are simple and usually focus on access control system in real domains such as military, enterprise, government, etc. We will survey in this section on these early access control models.

2.1.1 Discretionary And Mandatory Access Control

A system is said to have Discretionary Access Control (DAC) [1] if the creator or owner of an object can fully control the access to the object. A typical example is the file management under Linux, where the creator of the file has the full access to the file and also the right to assign to the other users any access to the file. In contrast, a system is said to have Mandatory Access Control (MAC[1], also known as 'non-discretionary access control' in security literatures) if the system administrator, instead of the owner, takes charge of the full access control.

In a MAC based system, the security policy is centrally controlled by a security policy administrator; users do not have the ability to override the policy, for example, an operation to grant access to files would be restricted. However, DAC governs the ability of subjects to access objects, but allows users to make policy decisions and/or assign security attributes. A classic Unix system of users, groups, and rwx (for *read, write* and *execute*) permissions is an example of DAC. MAC-based systems allow policy administrators to implement organization-wide security policies, unlike in DAC, users in MAC cannot override or modify a policy, either accidentally or intentionally. This allows security administrators to define a central policy that is secured (in principle) to be enforced for all users.

Generally speaking, MAC is useful rather in large access control systems with centralized access control servers and administrators. DAC is more flexible and suitable for small solutions where the owner determines the access control policies. However, currently on the web, there is no big difference between the owner and the administrator as the advancing information technologies lowers the threshold of technical operations. For example, on Flickr, the photo owner is the one in charge of control policy definition and she is able to do so because to define a policy is just to cross some check boxes on a specific web page for access control.

2.1.2 Access Matrix

Access Matrix model (AM) is introduced by B.Lampson [10] as one of the earliest access control models. It is a very general description of operating

Table 2.1: Example of Access Control Matrix

RIGHT	**server1**	**file2**	**device3**	**page4**
userA	connect	read,write	write	read
processB	connect	execute	read	update

system protection mechanism.

A security system based on AM model consists of a set of objects O, which is the set of resources that need to be protected (e.g., pages, files) and a set of subjects S, that consists of all active users (e.g., sessions, processes). There exists a set of rights R of the form $r(s,o)$, where $s \in S$, $o \in O$ and $r(s,o) \in R$. A right thereby specifies the kind of operation a subject is allowed to perform on an object. So the pairs build a matrix with the column as O and rows as S (or vise versa) and the contents of the matrix show possible right R as is shown in Table 2.1.

Because AM model does not define the granularity of protection mechanisms, it can be used as a model of the static access permissions in any type of access control systems. Implementation of AM for the current web scenarios as a two-dimensional array would suffer excessive memory costs.

2.1.3 Access Control List

In computer security, an access control list (ACL) is a list of (subject, operation) pairs attached to an object as shown in Figure 2.1. It is not tractable who invents the model but the ACL model has been implemented in many applications such as Microsoft Windows, Unix-like and Mac OS X operating systems. In Dec 2000, it becomes part of the 4th version of Network File System (NFS4) [76].

An access control list specifies who is allowed to access the object and what operations are allowed to be performed on the object. In an ACL-based security system, when a subject requests to perform an operation on

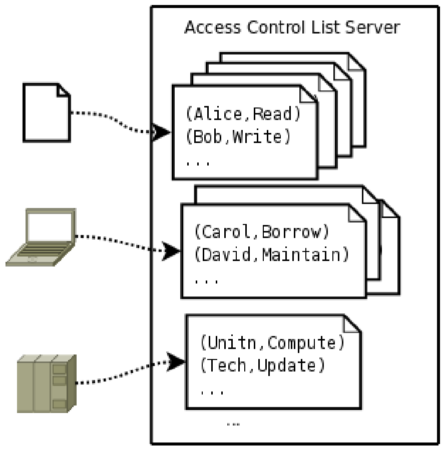
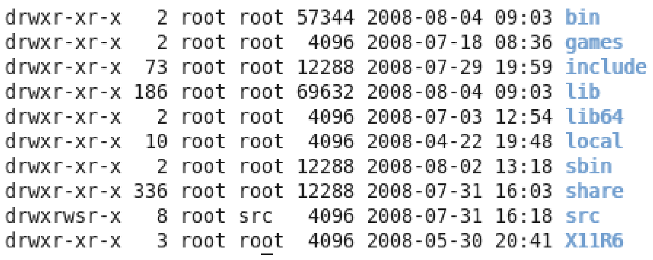


Figure 2.1: ACL Model



```
drwxr-xr-x   2 root root 57344 2008-08-04 09:03 bin
drwxr-xr-x   2 root root  4096 2008-07-18 08:36 games
drwxr-xr-x  73 root root 12288 2008-07-29 19:59 include
drwxr-xr-x 186 root root 69632 2008-08-04 09:03 lib
drwxr-xr-x   2 root root  4096 2008-07-03 12:54 lib64
drwxr-xr-x  10 root root  4096 2008-04-22 19:48 local
drwxr-xr-x   2 root root 12288 2008-08-02 13:18 sbin
drwxr-xr-x 336 root root 12288 2008-07-31 16:03 share
drwxrwsr-x   8 root src    4096 2008-07-31 16:18 src
drwxr-xr-x   3 root root  4096 2008-05-30 20:41 X11R6
```

Figure 2.2: ACL in a Linux Operation System

an object, the system first checks the list for an applicable entry in order to decide whether or not to proceed with the operation.

The ACL model is widely used in file management of different operating systems such as Microsoft Windows, OpenVMS, Linux and Mac OS X. Figure 2.2 shows a typical ACL in a linux OS, Ubuntu. The 9 digits at the beginning of each line specify the access rights (*read*, *write* and *execute*) for different users (*owner*, *group* and *other*).

A user identifier (UID) is the principal identifier of a subject. In file systems, the ACL is a data structure, usually a table, containing entries (known as access control entries ACE) that specify the rights of individual users or groups to specific system objects, such as a program, a process, or a file. Each accessible object contains an identifier to its ACL. The privileges or permissions determine specific access, such as whether a user can read from, write to, or execute an object. In some implementations an ACE can control whether or not a user, or group of users, may alter the ACL on an object.

In contrast to the AM model, ACL saves the space of the empty entries in the access control matrix.

2.2 Role-Based Access Control

Role-Based Access Control (RBAC) is proposed in [28] by D.Ferraiolo and R.Kuhn as a new generation of access control mechanism in contrast to Access Control Matrix (AM) and Access Control List (ACL). RBAC introduces a new concept: role as the intermediate between user and permission. In RABC, a permission specifies the ability to perform certain function on certain object with a (*operation, object*) pair. Now, RBAC has been adopted as an ANSI/INCITS standard [29]. It is a predominant model for access control in many fields such as commercial enterprises and governments.

The core concept of RBAC [28, 29] is *role*, which is defined by a set of permissions or a set of responsibilities that the user in this role can perform. Similarly to a role (or position) in the real world, a RBAC role relates to some *permissions*[1] to perform some operations on some objects. Users are not directly granted to any right as in AM or ACL, but have to activate a role that has already been assigned to him/her.

This greatly simplifies the management of permissions because a predefined role is rather static to the permissions. A user can be assigned to more than one role, but not necessarily all the assigned roles are activated to perform a certain task. R.Sandhu et al. introduced in [74] the concept of session into the RBAC model. A session is regarded a set of assigned roles activated for certain tasks. With control on user-role assignment, it is easy to restrict a user to behave as two disjoint roles such as the teller and the auditor in a bank scenario. This restriction can be performed by the session management at run time so that the role teller and auditor cannot be activated at the same time for the same user. *Role hierarchy* and related constraints are introduced as a feature of RBAC that permissions

[1]To be differentiated with *RelBAC* permission.

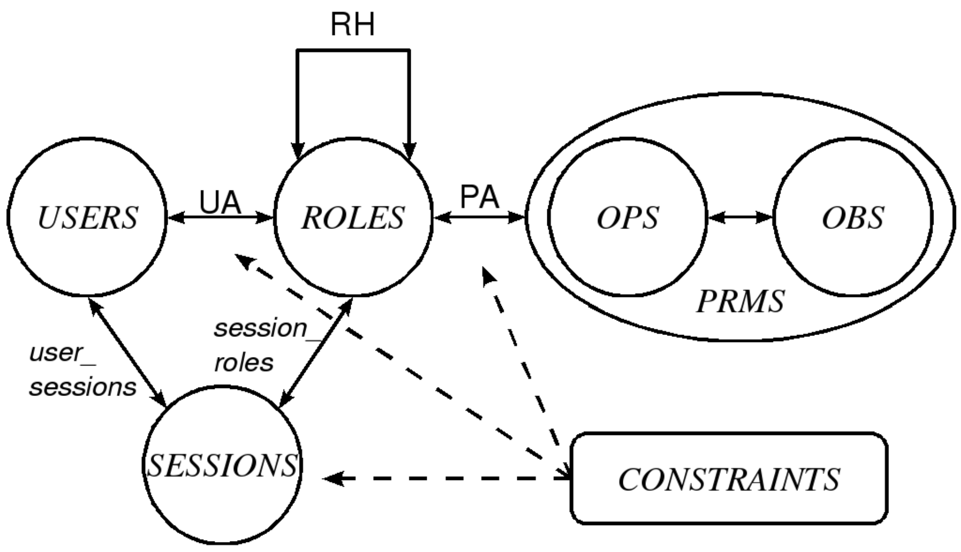


Figure 2.3: RBAC Model

may propagate through the role hierarchies.

We refer to the RBAC model [29] in Figure 2.3 as the standard and study the constraints, administrations and some extensions of RBAC in the following subsections.

2.2.1 The RBAC Model

As shown in Figure 2.3, the core of RBAC [29] is the definition of role, which originates from social positions such as the manager in an enterprise solution. The organism structures are relatively static and can be predefined. The advantage of role predefinition is that a permission assignment to an individual user is simplified to user-role membership declaration and the corresponding permission-role assignment has been pre-designed before. Thus, user-object connection is 'broken' by the role as a mediator. At run time, user-permission assignment is linear to the roles rather than to the Cartesian product of the operations and the objects.

The RBAC model has the following unary components:

- $USERS$ the set of all subjects;

- $ROLES$ the set of all roles;
- $OPERATIONS$ (OPS) the set of all operations;
- $OBJECTS$ (OBS) the set of all objects;
- $SESSIONS$ the set of all sessions;

The following binary components of RBAC:

- $PERMISSIONS$ ($PRMS$) the set of all pairs as $OPS \times OBS$;
- $USER\ ASSIGNMENTS$ (UA) the set of all pairs as $USERS \times ROLE$;
- $ROLE\ ASSIGNMENTS$ (RA) the set of all pairs as $ROLES \times PRMS$;
- $ROLE\ HIERARCHY$ (RH) the set of all pairs (r_1, r_2) as $ROLES \times ROLES$ representing a partial order $\succeq$ where $r_1 \succeq r_2$ only if all permissions of r_2 are also permissions of r_1 and all user of r_1 are also user of r_2;
- $user\_sessions$ the set of functions as $u : USERS \rightarrow 2^{SESSIONS}$;
- $session\_roles$ the set of functions as $s : SESSIONS \rightarrow 2^{ROLES}$;
- $authorized\_users$ the set of functions as $r : ROLES \rightarrow 2^{USERS}$ all users assigned to r and all r' that $r' \succeq r$.

The advantages of the RBAC model are due to the separation of users from the permissions. The roles serve as a mediator between the users and the permissions such that the access rights are granted directly to roles and a user can get a permission only by activating a corresponding role (that she is allowed to activate). This feature makes it possible to define the role hierarchy and grant to the roles appropriate access at design time; and only manage the correspondence between users and roles at run time. Thus the complexity of management task reduces from $USERS \times OPS \times OBS$ to $USERS \times ROLES$.

2.2.2 Separation of Duties

Separation of duties (SoD) is a principle well-known in financial accounting systems in which for example, no one should be assigned the duties to deposit cash and to reconcile bank statements. In general, the principle enforces conflict of interest policies, as a result of a user authorized of conflicting permissions. If a sensitive task consists of two steps, this requirement states that a different user should perform each step. More generally, when a sensitive task is comprised of n steps, a SoD policy requires the cooperation of at least k $(2 \leq k \leq n)$ different users to complete the task.

In [29] D.Ferraiolo, R.Sandhu et al. distinguish static SoD with dynamic SoD according to the time this constraint is enforced. Static separation of duties (SSD) enforces constraints on the assignment of users to roles at design time. Membership in one role may prevent a user from membership of one or more other roles. With the role hierarchy, inherited roles should be considered for this kind of prevention. Dynamic separation of duties (DSD) differs from SSD by placing constraints on the roles that can be activated within or across a user's session(s) at run time. SoD in RBAC is achieved by designing policies to specify mutually exclusive roles (for SSD) or mutually exclusive activated roles (for DSD).

2.2.3 Administration

Administration tasks of RBAC model was addressed in [29] and carefully studied in [23]. It consists many detailed tasks as listed in Table 2.2 clas-

Table 2.2: Administration Functions for RBAC

Administrative	System Support	Review
Add/Delete User	Create Session	Assigned Users
Add/Delete Role	Add Active Role	Role Permissions
Assign/De-assign User	Check Access	Session Roles
Grant/Revoke Permission	Delete Session	Assigned Roles
Add/Delete Inheritance	Drop Active Role	User Permissions
Add Ascendant	...	Session Permissions
Add Descendant		Authorized Users
...		Authorized Roles
		...

sified in the following three categories:

Administrative Functions Creation and maintenance of components such as adding a new user to the model. Additional hierarchical administrative functions for hierarchical RBAC model such as to add inheritance.

Supporting System Functions Implementation to support model constructs such as session management and session-role activation management.

Review Functions Review the results of administrative functions such as to get the permissions of a user or get the assigned users to a role.

2.2.4 Extensions

With the study of RBAC, many extensions have been proposed. Moyer et al. extend RBAC with a generalized model GRABC [65] in which environment and object roles are used to enhance expressiveness.

In [17], Chae et al. extend RBAC to support object hierarchy such that the efficiency brought by role hierarchy can be applied.

Time related extensions are proposed by E.Bertino et al. in [8]. J.Joshi et al. generalize them into [55, 54] as the GTRBAC model which allows expressing periodic as well as duration constraints on roles, user-role assignments, and role-permission assignments.

Purposed Based Access Control (PBAC) model is proposed in [16] by J.Byun et al. The PBAC model is designed for relational database where purpose information is associated with a given data element to specify the intended use of the data. They evaluate the ways to verify purposes and address the problem of how to determine the purpose of a given user when she access certain data.

2.3 Trust-Based Access Control

Another important access control model simulates the real world human relationships with the concept of *trust* (or *reputation*). Therefore, we call this kind of access control models as Trust-Based Access Control (TBAC). The trust is usually formalized as a quantified variable with a gossip based algorithm. In a TBAC system, the trust value is the standard for control decision. Once the value is above the threshold predefined, all access requests of the user are accepted; otherwise, all denied.

An example of successful reputation management is the online auction portal eBay [24]. In the reputation system of eBay, buyers and sellers can rate each other after each transaction, and the overall reputation of a participant is the sum of these ratings over the last 6 months. This system relies on a centralized system to store and manage these ratings.

The trust value lies on the behavior metrics such as follows.

Honesty The service claimed is coherent to the behavior.

Quality The service quality should be as expected.

Reliability The service should be available to some extend.

A basic TBAC model assumes a centralized trust server which knows about all peers, manages all the trust values of each peer according to the standards above and provides reliable answers to trust queries about peers. In the early days of online communities, some portals such as web of trust [68] and friend of a friend [67] provide the social networks for the Internet users together with the trust networks.

With the online community increasing enormously nowadays, fully P2P networks become popular. The Social Network applications mentioned in Section 1.3 are real examples of online communities.

Existing reputation systems on communities lie in two categories: use one's previous experience with an agent to estimate its reliability [53]; or use small-world phenomena to build chains of acquaintance to find other agents who can vouch for the reputation of a peer [46, 86]. In a full P2P scenario, trust calculation cannot rely on a centralized trust server. The gossip algorithm emerges as a simulation of the real world human relations. The trust can be accumulated not only by users directly interacted with, but also indirectly connected ones.

The preliminary gossip algorithm in Algorithm 2 may not terminate in an infinite network because on Line 7 there is a recursive call for the algorithm such that each neighboring peer will calculate the trust with its own neighbors. The algorithm relies on the transitivity of trust and multiplies the trust of peers through the path to the target peer. Whether it is a good choice among other ways of using this path is not the scope of this thesis. We just show that this is a thread of research for access control on communities.

The main feature of a trust-based system is the '*once for all*' authentication. Every coin has two sides. TBAC is good because all the control relies on a single standard, the trust only. Once the trust value reaches

Algorithm 2: Gossip

```
Data: peer: struct of {id: int; trust: float; neighbors: peer[ ];}
Input: peer current, peer target
Output: float target.trust
int i = 0;
target.trust = 0;
while i<current.neighbors.length do
    if target.id = current.neighbors[i].id then
        return current.neighbors[i].trust;
    else
        target.trust+=current.neighbors[i].trust*Gossip(current.neighbors[i],target);
    end
    i = i + 1;
end
return target.trust;
```

the threshold, all the access rights are granted and all requests are accepted. This is efficient with accuracy or granularity sacrificed as a trade-off. TBAC is too coarse grained upon the assignment because permissions are either all accepted or all denied without anything in between.

2.4 Formal Methods for Access Control Models

New access control models emerge for new challenges. The more complicated access control systems implemented, the more management tasks for the administration. Nowadays, it is hard to handle the access control management tasks by hand. Computer aided tools are proposed to help design, manage and implement the access control models.

We classify these tools into two categories, *non-logical* formalisms such as set theory, entity relationship diagram, XML etc. and *logical* formalisms such as algebras, First Order Logic and Description Logics. A survey will be shown in this section on how these tools are used on AC.

2.4.1 Non-Logical

Set theory, as we talk about, is a mathematical tool to formalize membership relations between individuals and their collections. It is used by R.Sandhu et al. to formalize RBAC in [74]. In Section 2.2, we have shown how to apply the set theory on the RBAC model. A role is modeled as a set and a user becomes an element of the set by the user-role assignment.

Set theory can be regarded the basic knowledge representation in Information science. Other theories such as different logics are all based on set theory (with additional semantics).

Entity Relationship (ER) model [19] is another powerful tool. It is specially popular for an abstract and conceptual representation of data. As we see from Section 1.1, the basic access control problem is a representation of the relations among subject, operation and object, as described by the ER model in Figure 2.4. The problem can be modeled as a 3-ary relation among the subject, object and operation.

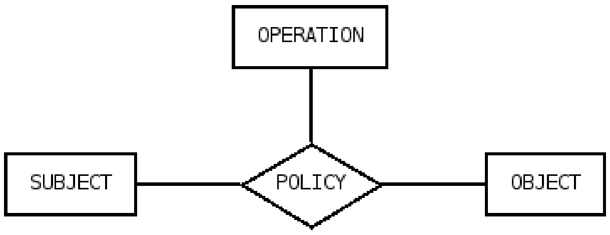


Figure 2.4: The ER Diagram of Basic Access Control

Yet another representation tool is the XML (eXtensible Markup Language) [84]. An XML based language, XACML (eXtensible Access Control Markup Language) [25] for representing knowledge about access control is approved as an OASIS [32] (Organization for the Advancement of Structured Information Standards) standard in Feb 2003. In February of 2005, XACML version 2.0 is approved as an OASIS Standard, along with some profiles including 'Core and Hierarchical Role Based Access Control

(RBAC)'. It provides a very flexible language to express access control that takes use of virtually all sorts of information as the basis for decisions. It is a functional superset of other popular access control schemes such as ACL, RBAC, etc. The language is extremely flexible with the XML syntax. Although it does not offer clear semantics, to use a web language to express access control is a good attempt.

2.4.2 Logical

With the evolving complexity of access control, the demand increases for automated administration tools. As formal logics have been used as the main method to achieve automated reasoning for computer aided tools, researchers made attempts to take use of logics to represent and reason about the access control domain.

To represent and reason about the specific access control models list in previous sections, different logical formalisms are studied e.g., logic algebra [2, 61, 13, 21], logic programming [9], Modal Logic [56, 18], Description Logic [89, 17], etc.

Logical Algebra

M.Abadi et al. study some of the concepts, protocols and algorithms for access control in distributed systems from a logical point of view and provide in [2] a logical language for ACL model together with some theories for deciding whether a given request should be granted. They focus on some practical aspects of distributed access control such as certification and delegation. However, the ACL model in their proposal has to face the fast complexity increase because of the various web resources. Moreover, the algebra they propose has to be accompanied with extra reasoning algorithms that are not supported by general logic reasoners.

F.Massacci presents a logical algebra for RBAC in [61]. The algebra relies on a tableaux method for reasoning which comes from the previous work of Modal Logic and Intuitionistic Logic. Now this method has been well studied and applied on Description Logic reasoners. The work shows a direction to explore the access control problem in Description Logics.

K.Broda et al. propose a Compiled Labeled Deductive System, called ACCLDS [14], for reasoning about role-based access control in distributed systems, which is built upon Massacci's tableaux based system. ACCLDS overcomes some of the limitations of of the system in [61] by combining its multi-modal propositional language with a labeling algebra that allows reasoning explicitly about dynamic properties of the accessibility relations. This facilitates a sound, complete, and more natural reasoning mechanism than [61]. They have also studied the limitations of the usefulness and their relation to the initial attempt in [2]. Their solution is based on First Order Logic and referred to a public theorem prover OTTER [62].

The HP lab devotes in research of access control and provides in [21] an account of fundamental situations in distributed systems using a resource-based process calculus and a hybrid of Hennessy-Milner (a Temporal Logic) and resource logic (a Modal Logic). This yields a consistent account of operational behavior and logical reasoning for access control, that includes an analysis of co-signing, roles and chains-of-trust as has been stressed in [2]. The HP proposal focuses on real application domains and the hybrid of algebra and logic makes the system realistic for the concrete access control problems. This tells us that real world problems might not be solved with one logical tool such as a well studied algebra or logical framework, but rather a hybrid solution that uses advantages of different frameworks might achieve the goal.

Logic Programming

For logic programming approaches, Prolog is the most widely used language. E.Bertino et al. provide a logical framework [9] for reasoning about discretionary, mandatory and role-based access control models based on C-Datalog [47] which is an object-oriented variation of Datalog [80, 81]. Each instance of the proposed framework corresponds to a C-Datalog program, interpreted according to a stable model semantics.

The use of logic programming on access control presentation has the advantage to formalize the state transitions. This thread of work is not the scope of this thesis.

Description Logic

Description Logics [4] (a set of logical frameworks with different expressiveness) arouse the interests of computer science scholars by their expressiveness and decidability. Many literatures [89, 17, 30] attempt to use Description Logic to formalize access control models and use state of the art Description Logic reasoners to prove security properties.

C.Zhao et al. in [89] use description logic to represent RBAC. In their proposal, *users, roles, sessions* and *permissions* are formalized as DL concepts but *objects* are regarded encapsulated inside *permissions* together with *operations*. This results in explosion of the permission and the difficulty to specify policies about *objects*. Moreover, they propose to use only *existential restriction* of DL for *permission assignments*. This does not make full use of the powerful expressiveness of DL. Anyway their proposal is an early attempt and inspiring in the use of DL on access control.

Another formalization with description logic is proposed by J.Chae et al. to formalize RBAC with DL in [17], where an *operation* is represented by a DL role. But their system has several flaws as follows.

- Miss use of *existential quantifier*. In the semantics of their formalization, the assignment with formula '$Admin \sqsubseteq \exists CanRead.Log$' assigns the read access to all the administrators onto all the log files. But classic DL semantics of this formula enforces only *some* connections from administrators to the log files but not *all*.
- Awkward use of unnecessary DL roles such as 'assign' and 'classify'. These DL roles are supposed to connect users to RBAC roles or object to object classes but ABox mechanism of DL serves well in this purpose.

However, their work is still meaningful to us in the following ways.

- The way they formalized *operation* is inspiring as it is natural to represent an action (of the operation from a subject to an object) with a binary relation rather than a unary concept.
- They extend RBAC with the object hierarchy similar to the user hierarchy which facilitates the permission propagation.

Recently, T.Finin et al. propose to use OWL [69] as the formalization of the RBAC model in [30]. They provide two ways to formalize a RBAC role, as class or as attribute. N3Logic is used together with DL subsumption reasoning. Authorization decision queries can be answered using DL reasoners in their system. Their work is a variation of applying DL on RBAC representation, and OWL can be a candidate language we implement our model.

2.5 Summary

For the general access control problem described in Chapter 1, several access control models are proposed for different domains such as military or commercial solutions. AM is one of the early attempts and applies in those early access control systems; ACL is prevailing in operation systems and still active in some current solutions because of its simplicity; MAC is suitable for military use while DAC serves well in urban domains such as commercial enterprises; RBAC is popular specially for large commercial solutions but not restricted to; etc. When the new challenges come with the new Internet technologies as described in Chapter 1, these models are no longer suitable.

However, they are inspiring in the following aspects:

- AM is old, but has the advantage to control the access in both the subject and the object point of view, although it suffers the sparse matrix and waste of space.

- RBAC is prevailing currently, but focuses on static predefined role hierarchies which is not suitable for the highly dynamic state of the art web scenarios. Yet the idea of separate users from direct assignment to the permissions is wonderful and we can borrow this idea to our new model.

- TBAC is new in nowadays networks and especially successful (up to now) for peer to peer architectures. We may take it as an interesting attribute that can be supported by our new model.

We will show in the next chapter how we build our model considering these existing models, facing those challenges mentioned in Chapter 1.

Chapter 3

RelBAC Model

In this chapter, we build a novel access control model: Relation-Based Access Control, shorted as *RelBAC*. It is a new model for the Information Age to face the new challenges as discussed in Chapter 1. The main novelty of this model is that a permission with the intuition as the operation that a subject is allowed to perform on an object, is modeled as a binary relation between subject and object. Thus, it is decoupled from the subject and the object, and treated as a first class component of the model.

The *RelBAC* model has at least the following advantages.

- Permissions are decoupled from subjects or objects so that we can model the dynamics of permission evolution easily.

- Subjects and objects in *RelBAC* can be classified into complex (typically hierarchical tree-like) structures such as business organisms or online social communities. No matter these structures are relatively stable or rather vulnerable, *RelBAC* offers a way to model these evolving hierarchies.

- The *RelBAC* model provides fine-grained access control with different granularity such as *one, some, minimum m, maximum n, exactly k* and *all* from both perspectives of subjects and objects.

- Based on an Entity Relationship Model, *RelBAC* can be easily designed and integrated into any large software systems. Moreover, this brings great extensibility to capture new attributes, contexts, etc.

- Formalized with a Description Logic based logical framework, *RelBAC* offers compact representation of access control rules with clear semantics and cardinality expressiveness, in addition to reasoning ability that facilitates access control management and security property analysis.

The rest of the chapter is organized as follows: Section 3.1 describes the *RelBAC* model with an ER diagram; Section 3.2 shows the Description Logic based logical framework for *RelBAC* and discusses in details how to build access control rules in this framework. Section 3.7 summarizes the correspondence between fine-grained access control assignments and *RelBAC* rules.

3.1 ER Model of *RelBAC*

The *RelBAC* model proposed in [44] has evolved into a more compact form with the removal of unimportant parts. Basic components of the *RelBAC* model are illustrated with the Entity Relationship (ER) diagram in Figure 3.1. According to the standard format of ER diagrams [19], we have `SUBJECT, OBJECT` as entity sets and `PERMISSION` as relations. Here we get into details of each component.

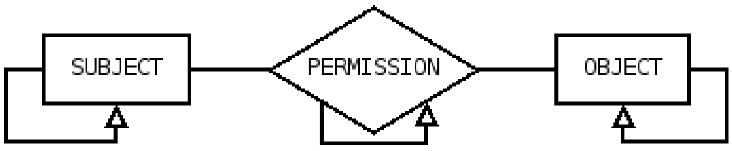


Figure 3.1: The ER Diagram of the *RelBAC* Model

SUBJECT: a set of subjects that intend to access some resources. The loop on SUBJECT represents the 'IS-A' relation between sets of subjects. The largest subject set is the collection of all the subjects concerned (e.g., all people of University of Trento).

OBJECT: a set of objects or resources that subjects intend to access. The loop on OBJECT represents again an 'IS-A' relation between sets of objects. The largest object set is the collection of all the possible objects of the system (e.g., anything with a URI).

PERMISSION: the intuition is that a permission is an operation that subjects can perform on objects. To capture this, a permission is named with the name of the operation it refers to (e.g., *Read* or *Update*). A PERMISSION is a relation between SUBJECT and OBJECT, namely a set of (subject, object) pairs (e.g., $Update(Ilya, Code1.0)$). The loop on PERMISSION represents the 'IS-A' relation between permissions.

RULE (short for access control rule): a RULE associates a PERMISSION to a specific set of (SUBJECT,OBJECT) pairs by two means. First is to state that a set of SUBJECTs is a subset of those that can access a given set of OBJECTs with a given PERMISSION, and we call it a *subject-centric* RULE; the other way is to state that a set of OBJECTs is a subset of those that can be accessed by a given set of SUBJECTs with given PERMISSION, which we call *object-centric* RULE. These two kinds of rules offer us different views of the access control system such that we can define rules from either the subject or the object stand.

For example, in Linux system, access rights are attached to files. Given a file, this makes it easy to check the accessibility of a user to the file. But given a user it is hard to get all the objects that can be accessed by the user or to grant some given user the access rights to specific objects. *RelBAC* offers the *object-centric* rule which makes this as easy as to look from the object column in an access control matrix [10]. Moreover, in

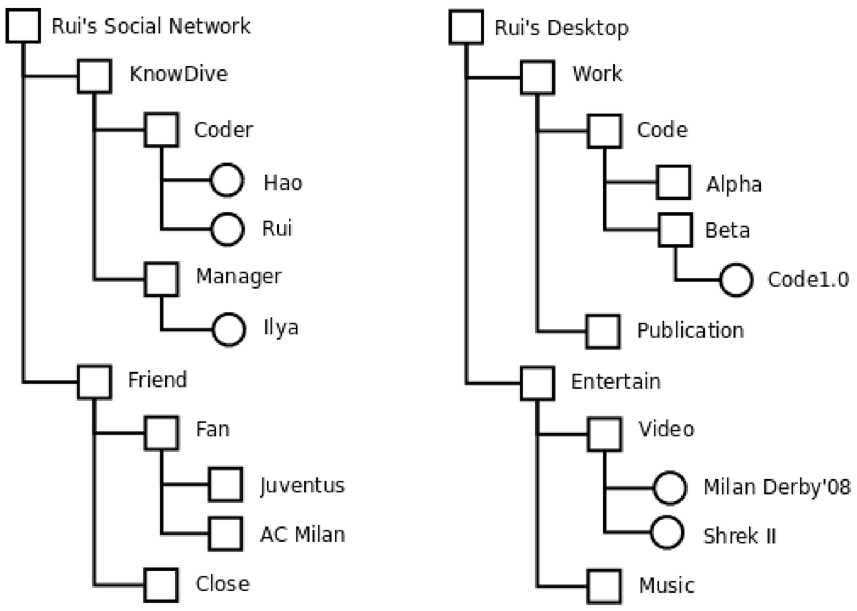


Figure 3.2: `SUBJECT` and `OBJECT` Hierarchies

addition to the inner relations among subjects or objects, we can even declare the relation among permissions without mentioning the subject or object involved. This makes the permission, formerly an association tightly banded to subject or object such as in RBAC [29], now a first class entity in the system. Thus, context information such as system load or connection number in a pervasive environment could be encoded into specific permissions.

As an example of *RelBAC*, we have an access control problem in the scenario of Social Desktop [75]. In Figure 3.2 and 3.3, we show examples of `SUBJECT`, `OBJECT` and `PERMISSION` hierarchies respectively. The squares represent sets and circles represent individuals (e.g., 'Friend' and 'Code' are subject and object sets respectively; 'Hao' and 'Shrek II' are individual subject and object). Similarly, squares in Figure 3.3 represent sets of permissions while circles represent individual permissions which are (subject, object) pairs (e.g., '(Rui, code1.0)' is an individual permission of 'Read'). The lines connecting items represent the 'IS-A' relationship.

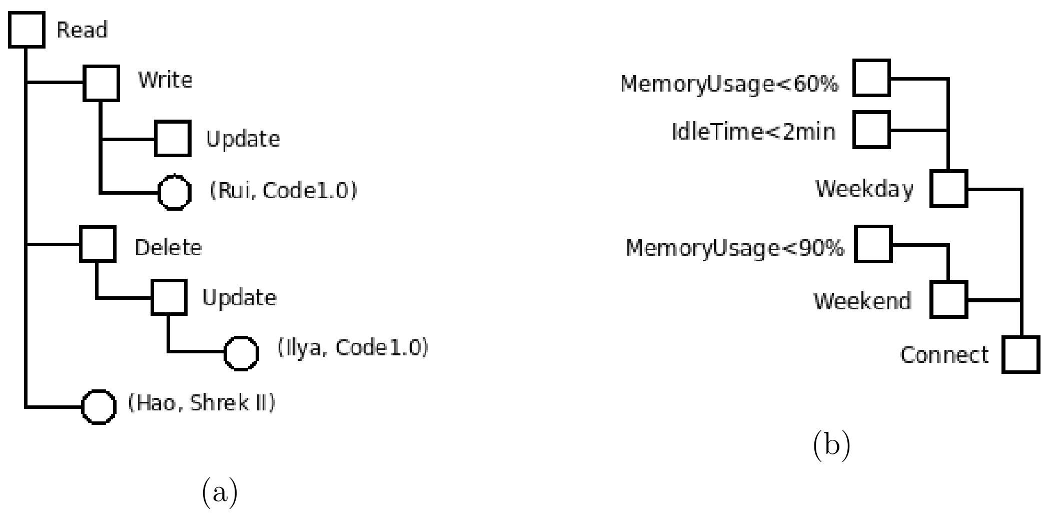


Figure 3.3: `PERMISSION` Hierarchies

Thus in detail, Rui's social network is partly shown as the left of Figure 3.2 where he has mainly two groups of people, one from KnowDive research team and the other as friends. In the research team Hao is a coder, and Ilya is a team manager. There is a subgroup of soccer fans in friends with more specific subgroups supporting different soccer teams 'Juventus' and 'AC Milan' (two professional soccer teams in Italian Serie A). Rui has also some specific close friends in the network. For desktop resources, Rui's working stuffs such as codes and publications are organized in a resource tree together with files for entertainment. The subject and object hierarchies follow the intuition that the lower in the hierarchy, the more specific items are collected.

Now something more interesting comes! Permissions may take the from of a tree or even a directed acyclic graph (DAG). Here in Figure 3.3-a we state that to write and to delete a file are two more specific (in the sense less pairs as instances) operations than to read; to update is even more powerful (with less pairs) than to write or to delete. The circle denoting a pair $(Ilya, Code1.0)$ under 'Update' square represents that Ilya can update the Code1.0 and the pair at bottom of Figure 3.3-a tells us that Rui can write

Code1.0 and Hao can read (view) the video named 'Shrek II'. Figure 3.3-b shows that permissions may take the form of an inverted tree. Contexts such as accessing time and system load are important standards for access control and we can have various permissions defined accordingly. These permission hierarchies might be different even for the same environment by different administrators, e.g., one may consider system load first and another emphasizes on accessing time instead. To be precise, Figure 3.3-a is a DAG when we take the two square labeled 'Update' the same. Then, Ilya can not only update, delete and read Code1.0, but can also perform write access on it.

As we mentioned in Section 2.4, Entity Relationship (ER) Model is an abstract and conceptual representation of data. It is widely used in software engineering to model a relational schema, and produce a conceptual schema or semantic data model of a system. Therefore, an ER model has the advantage to be easily integrated into system design. Also, as it is adaptive in almost all application domains (no matter rural or military, commercial or educational), *RelBAC* inherits the adaptiveness to be able to model the access control in a wide variety of domains.

Let us see the example of the Semantic Desktop scenario at the beginning of this section. The two trees in the Figure 3.2 describe part of Rui's social network and semantic desktop resources (as we cannot show all his social network and resources). In addition, as shown in Figure 3.3, permissions may form complex structures too.

The information can be collected as the following:

User : {Rui, Hao, Ilya,...}

User Group : {Rui's Social Network, KnowDive,...}

Resource : {code1.0, Milan Derby 2008, Shrek II,...}

Resource Class : {Rui's Semantic Desktop, Work, Entertain,...}

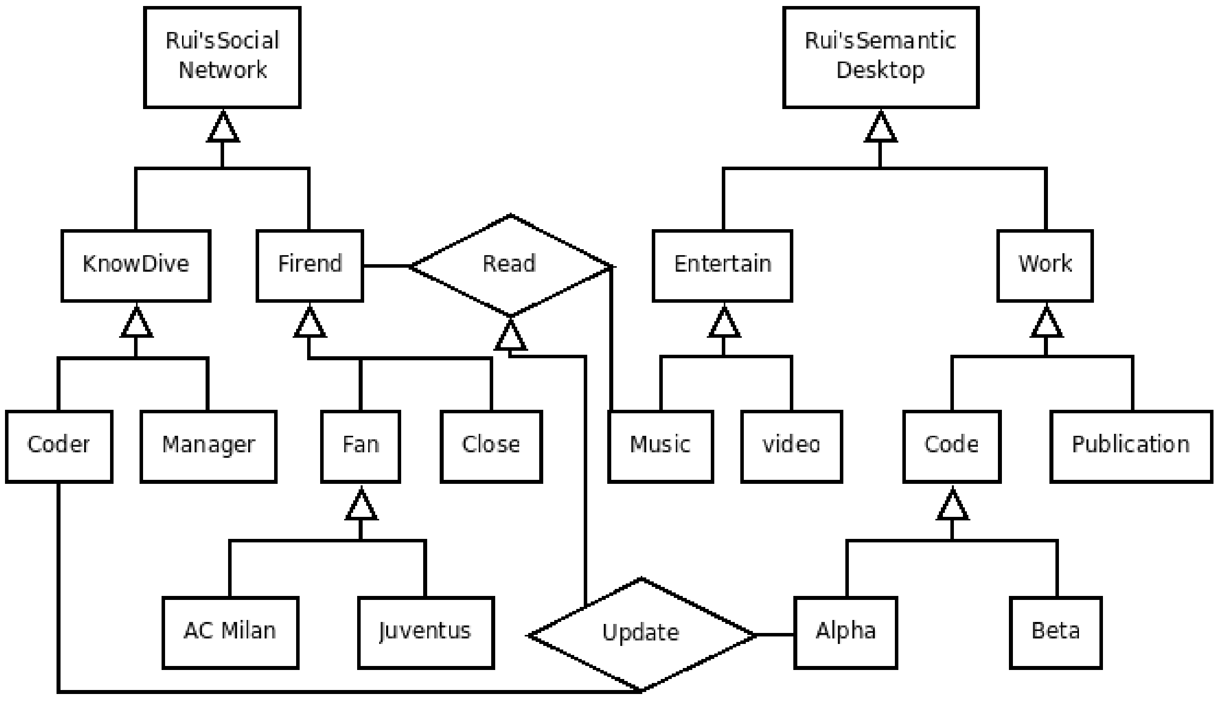


Figure 3.4: ER Model of the Social Desktop Scenario

Operation : {Read, Write, Update,...}

Policy : {'All friends are allowed to read musics', 'coders are allowed to update Alpha codes', 'a close friend is also a friend', 'those allowed to update are also allowed to read the same object',...}

Here we use the word 'Policy' in a broad sense for all the entries in the knowledge base of an access control system, including not only the access control rules but also the description of systems such as the 'IS-A' relation between groups, etc.

As an instantiation of *RelBAC* shown in Figure 3.1, we can represent the hierarchies of the Social Desktop scenario with the ER diagrams in Figure 3.4. Rui's social network and desktop resources may change frequently, while the ER diagram can be extended to capture these changes.

3.2 Logical Framework of *RelBAC*

Inspired by the practices in Section 2.4.2, Description Logic stands out for *RelBAC* representation. It has at least the following advantages: *expressiveness, decidability, state of the art reasoners* and *coherence with ER.*

- DL is a sub-set of First Order Logic, but still preserves much expressiveness such as *value restriction* which can be easily applied on the access control domain with compact syntax.

- DL is a decidable sub-set of First-Order Logic (FOL) and complexity problems corresponding to certain *constructors* have been studied by logicians for years.

- There are many off-the-shelf reasoners for automated reasoning of DL knowledge bases. A lot of scholars are working on building algorithms and reasoners for DL.

- With the original purpose to represent related database, DL and ER share some interests in common. It is easy to represent an ER model in DL.

Following the relationship between ER and DL as discussed in [4], the atomic elements of *RelBAC* are concepts, which intuitively can be thought of as names for sets of objects. We also have DL roles which, in turn, can be thought of as names for binary relations.

We use $\mathcal{ALCIOQ}(\neg)$ as the DL for *RelBAC*. Taking into account the necessary access control components, we have the following:

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

where $U_i(i = 1, ..., m)$ are concepts for users, 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 $Audit$. In this thesis, we use italic words that begin with capital letter(s) as concept and role names. The syntax and semantics of $RelBAC$ is clear according to $\mathcal{ALCIOQ}(\neg)$.

$RelBAC$ uses *subsumption* '$\sqsubseteq$' to represent partial order '$\geq$' relation among user groups, among object classes and among permissions. Thus, a 'generalization (IS-A)' relation is translated into a *subsumption* axiom. Then '$\geq$' helps to build inheritable hierarchies among subjects, among objects and among permissions.

$$U_i \geq U_j \qquad iff \qquad U_i \sqsubseteq U_j \tag{3.1}$$

$$O_i \geq O_j \qquad iff \qquad O_i \sqsubseteq O_j \tag{3.2}$$

$$P_i \geq P_j \qquad iff \qquad P_i \sqsubseteq P_j \tag{3.3}$$

3.3 General Rules

Rules in $RelBAC$ usually take the form of *inclusion* formulas. Given the concepts U, O and a permission P, a permission *assignment* policy has one of the following forms:

$$U \sqsubseteq \exists P.O \tag{3.4}$$

$$O \sqsubseteq \exists P^{-1}.U \tag{3.5}$$

$$U \sqsubseteq \forall P.O \tag{3.6}$$

$$O \sqsubseteq \forall P^{-1}.U \tag{3.7}$$

For example, to say that *all the close friends can download some music(s)*, we can use Rule 3.4 as

$$Friend \sqsubseteq \exists Download.Music;$$

to say that *all the music can be downloaded by some friend(s)*, we use Rule 3.5 as

$$Music \sqsubseteq \exists Download^{-1}.Friend;$$

to say that *the friends can download only the music(s)*, we use Rule 3.6 as

$$Friend \sqsubseteq \forall Download.Music;$$

to say that *the code can be read only by the KnowDive members*, we use Rule 3.7 as follows.

$$Code \sqsubseteq \forall Read^{-1}.KnowDive$$

Cardinality related rules can be expressed as:

$$U \sqsubseteq \geq nP.O \tag{3.8}$$
$$O \sqsubseteq \geq nP^{-1}.U \tag{3.9}$$
$$U \sqsubseteq \leq nP.O \tag{3.10}$$
$$O \sqsubseteq \leq nP^{-1}.U \tag{3.11}$$

For example, to say that *each KnowDive member should program for minimum 1 project code*, we use Rule 3.8

$$KnowDive \sqsubseteq \geq 1\ Program.Code;$$

to say that *each project code should be programed by maximum 2 KnowDive members*, we use Rule 3.11

$$Code \sqsubseteq \leq 2\ Program^{-1}.KnowDive.$$

Here, we use only two examples to illustrate the *minimum* value restriction formed *user-oriented* rule and the *maximum* value restriction formed *object-oriented* rule. The other two forms of cardinality related rules are similar.

In $RelBAC$, we do not need special rules to restrict access, which is commonly referred to as 'prohibition'. Because we can achieve this kind of restricts by saying that *a set of users is a subset of the complement set of those allowed to access something.* Or symmetrically, *a set of objects is a subset of the complement set of those allowed to be accessed by someone.* For example, *Coders are not allowed to update the publications* can be enforced as the following rule.

$$Coder \sqsubseteq \neg\exists Update.Publication$$

3.4 Rules Involving Instances

In DL, an instance related axiom is called an ABox and we find it also applicable in $RelBAC$. An instance of a user group, object class or permission describes a state of the access control system. In $RelBAC$, policies about instance are called *access control state* $\mathcal{S}$. We use words starting with lower case letter(s) to denote names of instances such as $hao, rui, shrek\_II$... With the names for concepts and roles, we can describe which concept an instance belongs to or which role a pair of instances belongs to.

For example, $KnowDive(rui)$ declares rui denoting the user 'Rui' is a KnowDive group member. $Video(shrek\_II)$ means that the file denoted by $shrek\_II$ belongs to the object class Video. $Download(hao, shrek\_II)$ declares the pair $(hao, shrek\_II)$ is an instance of $Download$ with the intuition that *Hao is allowed to download the video Shrek II.*

We can assign permissions to a collection of users with the *set* constructor such as to give $Update$ permission of $Code$ to hao and $ilya$. We don't have to assign it one by one, but with a single rule as

$$\{hao, ilya\} \sqsubseteq \exists Update.Code$$

Then with the *set* and *complement* we can easily revoke the permission from some particular users out of the user group such as *to allow all friends but not Hao to upload musics* with the rule as the following.

$$Friend \sqcap \neg\{hao\} \sqsubseteq \exists Upload.Music$$

With the *membership* and *fill* constructor, *RelBAC* can express *subject-oriented* rules as follows.

$$U \sqsubseteq P : o \tag{3.12}$$

$$(P : o)(u) \tag{3.13}$$

$$(\exists P.O)(u) \tag{3.14}$$

$$(\forall P.O)(u) \tag{3.15}$$

$$(\geq nP.O)(u) \tag{3.16}$$

$$(\leq nP.O)(u) \tag{3.17}$$

where u is an individual user and o is an object instance; U is a user group and O is an object class; P is a permission.

For example, to say that *close friends can download a video named Shrek II*, we use Rule 3.12 as

$$CloseFriend \sqsubseteq Download : shrek\_II;$$

to say that *Hao can download Shrek II*, we use Rule 3.13 as

$$(Download : shrek\_II)(hao);$$

to say that *Hao can update some codes*, we use Rule 3.14 as

$$\exists Update.Code(hao);$$

to say that *Hao can upload only videos*, we use Rule 3.15 as

$$\forall Upload.Video(hao);$$

to say that *Hao can update minimum 10 codes*, we use Rule 3.16 as

$$\geq 10 Update.Code(hao);$$

to say that *Hao can download maximum 15 videos*, we use Rule 3.17 as follows.

$$\leq 15 Download.Video(hao)$$

We can have corresponding *object-oriented* rules (3.18–3.23) respectively as the following. We do not show examples of these rules as they are similar to the *subject-oriented* rules above.

$$O \sqsubseteq P^{-1} : u \tag{3.18}$$

$$(P^{-1} : u)(o) \tag{3.19}$$

$$(\exists P^{-1}.U)(o) \tag{3.20}$$

$$(\forall P^{-1}.U)(o) \tag{3.21}$$

$$(\geq nP^{-1}.U)(o) \tag{3.22}$$

$$(\leq nP^{-1}.U)(o) \tag{3.23}$$

3.5 The 'All' Rule

Up to now, we have introduced 8 kinds of rules (3.4-3.11) which assign respectively to user the permission to access *some, only, minimum/maximum n* objects with respect to Rule 3.4, 3.6, 3.8/3.10 (or restrict object to be accessed by *some, only, minimum/maximum n* users with respect to assignment 3.5, 3.7, 3.9/3.11). But in a classic access control system, the assignment from a set of users to a set of objects usually means that the access has been granted to all the users onto all the objects. Therefore, we need a form of permission assignment that the given group of users can access *all* the objects in a given class.

Inspired by Alex Borgida[1] we come to the following theorem for the rule from all subjects onto all objects.

Theorem 1. *Let C be a concept of object, and P a permission, then $\forall C.P \equiv$ where $(\forall C.P)^{\mathcal{I}} = \{a | \forall b \; b \in C^{\mathcal{I}} \rightarrow (a,b) \in P^{\mathcal{I}}\}$.*

Proof.

$$\begin{aligned}
&(\forall C.P)^{\mathcal{I}} \\
=&\{a|\forall b \; b \in C^{\mathcal{I}} \rightarrow (a,b) \in P^{\mathcal{I}}\} \\
=&\{a|\forall b \; \neg b \in C^{\mathcal{I}} \vee (a,b) \in P^{\mathcal{I}}\} \\
=&\{a|\forall b \; \neg b \in C^{\mathcal{I}} \vee \neg\neg(a,b) \in P^{\mathcal{I}}\} \\
=&\{a|\forall b \; \neg(a,b) \in P^{\mathcal{I}} \rightarrow \neg b \in C^{\mathcal{I}}\} \\
=&(\forall \neg P.\neg C)^{\mathcal{I}}
\end{aligned}$$

□

Theorem 1 shows that to assign all the objects in a concept to user(s) can be achieved with a rule in which the permission role *negation* and the object concept *negation* are applied in *value restriction*. We can express in Rule 3.24 that all users in U can access all the objects in O with permission P; and in Rule 3.25 that all the objects in O can be accessed by all the users in U with P as follows. We name these new rules built with the short form proved in Theorem 1 as ‘All’ rules.

$$U \sqsubseteq \forall O.P \tag{3.24}$$

$$O \sqsubseteq \forall U.P^{-1} \tag{3.25}$$

[1]Alex Borgida http://www.cs.rutgers.edu/ borgida/

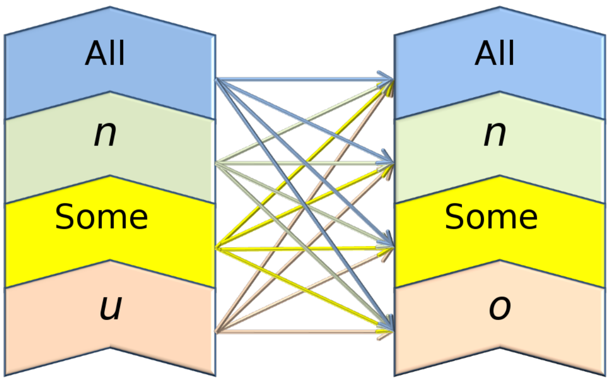


Figure 3.5: Fine Grained Access Control of *RelBAC*

and we can have formulas with instances:

$$(\forall O.P)(u) \tag{3.26}$$

$$(\forall U.P^{-1})(o) \tag{3.27}$$

For example, *project leaders can update all the code*, is expressed with Rule 3.24 as

$$ProjectLeader \sqsubseteq \forall Code.Update;$$

All the code can be updated by project leaders is expressed with Rule 3.25 as follows.

$$Code \sqsubseteq \forall Update^{-1}.ProjectLeader$$

The 4 rules (3.24–3.27) capture the common and only possible permission assignments in existing access control system such as RBAC[29]. Obviously we can express more with the other 8 kinds of assignments.

3.6 Grained Cardinality

To be exact, we have fine-grained access control on cardinality as is shown in Figure 3.5. Among the 16 combinations of the possible involved users

Table 3.1: *RelBAC* Rules Correspondence

NO.	Assignment	*RelBAC* Rule(s)
1	$u \rightarrow o$	$P(u,o) \vert P^{-1}(o,u) \vert (3.13) \vert (3.19)$
2	$u \rightarrow \mathit{some}\ O$	(3.14)
3	$u \rightarrow \mathit{minimum} \vert \mathit{maximum} \vert \mathit{exactly}\ n\ O$	$(3.16) \vert (3.17) \vert (3.16{,}3.17)$
4	$u \rightarrow \mathit{all}\ O$	(3.15)
5	$\mathit{some}\ U \rightarrow o$	(3.20)
6	$\mathit{some}\ U \rightarrow \mathit{some}\ O$	$\exists P.O \sqsubseteq U \vert \exists P^{-1}.U \sqsubseteq O$
7	$\mathit{some}\ U \rightarrow \mathit{minimum} \vert \mathit{maximum} \vert \mathit{exactly}\ n\ O$	$\geq nP.O \sqsubseteq U \vert \leq nP.O \sqsubseteq U \vert$ $\geq nP.O \sqsubseteq U, \leq nP.O \sqsubseteq U$
8	$\mathit{some}\ U \rightarrow \mathit{all}\ O$	(3.5)
9	$\mathit{minimum} \vert \mathit{maximum} \vert \mathit{exactly}\ m\ U \rightarrow o$	$(3.22) \vert (3.23) \vert (3.22{,}3.23)$
10	$\mathit{minimum} \vert \mathit{maximum} \vert \mathit{exactly}\ m\ U \rightarrow \mathit{some}\ O$	$\geq mP^{-1}.U \sqsubseteq O \vert \leq mP^{-1}.U \sqsubseteq O \vert$ $\geq mP^{-1}.U \sqsubseteq O, \leq mP^{-1}.U \sqsubseteq O$
11	$\mathit{minimum} \vert \mathit{maximum} \vert \mathit{exactly}\ m\ U \rightarrow \mathit{all}\ O$	$(3.9) \vert (3.11) \vert (3.9{,}3.11)$
12	$\mathit{all}\ U \rightarrow o$	(3.12)
13	$\mathit{all}\ U \rightarrow \mathit{some}\ O$	$\forall P^{-1}.U \sqsubseteq O$
14	$\mathit{all}\ U \rightarrow \mathit{minimum} \vert \mathit{maximum} \vert \mathit{exactly}\ n\ O$	$(3.8) \vert (3.10) \vert (3.8{,}3.10)$
15	$\mathit{all}\ U \rightarrow \mathit{all}\ O$	(3.6,3.7)

and objects for access control, *RelBAC* can deal with 15 of them except the assignment which implies that 'minimum/exactly/maximum m subjects in $U(m \leq |U|)$ are allowed to access minimum/exactly/maximum n objects in $O(n \leq |O|)$'. Here we use 'm' instead of 'n' for users to avoid ambiguity.

Here in Table 3.1, u, o, U, O and P represent respectively *an individual user, an object instance, a user group, an object class* and *a permission*. We list the rule(s) to be referred to in order to achieve the 15 types of permission assignments. Some are directly achieved by a *RelBAC* rule such as Assignment (1,2,4,5,8,12); some require the combination of two rules such as Assignment (15) and those related to 'exactly' cardinality; and some are not directly represented by *RelBAC* rules we introduced in previous sections such as Assignment (6,7,10,13) but we can still use DL *inclusion* axioms to achieve these assignments. So we extend the definition of `RULE` a bit to allow the destination concept to appear on either side of '$\sqsubseteq$' and then all of the assignments in Table 3.1 can be represented with *RelBAC* rules or rule combinations.

Actually, the 'supersumption' and 'subsumption' relations between sets can be both represented with *inclusion* axioms by just exchanging the position of the two sets in the axiom. 'Equivalence' can be also used to form rules with *equality* axioms in addition to 'subsumption' rules introduced in Section 3.3. However, it is more strict because it defines not only the 'right' but also the 'duty'. For example, in the following rule

$$Coder \equiv \exists Update.Code$$

the users in the 'Coder' group have been assigned the permission to update the objects in the 'Code' class. Moreover, at the same time, they are obliged to update some object in that class. Otherwise, they cannot be a member of 'Coder'. So we seldom use 'equivalence' formed rules in practice unless a formal definition is intended.

3.7 Summary

In this chapter, we introduced the *RelBAC* ER model and formalized *RelBAC* with Description Logic. We focused here on the representations of fine-grained access control with various of cardinalities and different forms of access control rules in forms of *inclusion* and *equality*.

In the next chapter, we will show in details the reasoning ability of *RelBAC*. Different access control tasks can be transformed to reasoning tasks for an off-the-shelf DL reasoner.

Chapter 4

Reasoning in *RelBAC*

The advantage of using a logical framework for an access control model is the automated reasoning ability of the logic. As discussed in Chapter 1, current access control systems are supposed to face many new challenges such as dynamic knowledge structure, various data types, evolving contexts, etc. These challenges bring us more and more complex management tasks, so computer aided administration tools are almost compulsory for a modern access control system.

We can identify two phases when we need reasoning. At design time, it serves as a support tool for policy writers to determine possible conflicts or to verify if the set of policies satisfy a desired static security property (i.e. separation of duties). We discuss the reasoning with *RelBAC* logic for these properties in Section 4.1. The reasoning abilities can be also used at run time, which we will show in Section 4.2, how to use reasoning in the implementation of *access control decision* which means to verify if an access request is permitted or rejected by the applicable policies.

Before starting the main part of this section, we introduce some naming traditions. In *RelBAC*, the knowledge base is divided into two parts. As ABox in DL, the knowledge about individual users or objects is called $\mathcal{S}$ which stands for states as it describes the state of the system. For example, the user membership to a group, access control policies about individual users etc. are all typical knowledge of $\mathcal{S}$. The other part deals with knowledge without any individual such as hierarchies of groups and classes and access rules about groups and classes. It includes TBox axioms as in DL and called $\mathcal{P}$ for access control Policy. For example, the rules (3.4-3.11) in Section 3.2 are typical knowledge in $\mathcal{P}$. In this thesis, we will follow the tradition that $\mathcal{S}$ is called the *state base*; $\mathcal{P}$ is called the *policy base* and putting the two bases together, we get the *knowledge base* of the *RelBAC* access control system.

4.1 Design Time Reasoning

To specify a desired access control system, security administrators have to carefully design policies that meet the access control requirements. When the number and/or the complexity of the policies is not trivial this process is error-prone. It is very useful to offer a support tool that can automatically detect possible conflicts among policies and/or verify that the set of specified policies satisfy some required security properties. *RelBAC* provides automated reasoning as such a tool at design time.

4.1.1 Hierarchy Management

A feature of *RelBAC* is its natural formalization of hierarchy as discussed in Section 3.2. The 'generalization' relations in *RelBAC* ER model build tree-like hierarchies among each of the three components: user groups, object classes and permissions.

RelBAC reasoning can help to build the hierarchies at design time. Here we introduce the *subsumption* reasoning task coherent to classic Description Logic [4].

Definition 2. Subsumption A concept C is subsumed by a concept D with respect to $\mathcal{P}$ if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ for every model $\mathcal{I}$ of $\mathcal{P}$. A subsumption checking is to check the following reasoning task.

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

There are many constraints in real life for the hierarchy in an access control system. Gavrila et al. have discussed some user/role and role/role relationship constraints with first order logic in [34]. It is much easier to describe these constraints in *RelBAC*. Here we list the three constraints about hierarchies in italic and discuss how *RelBAC* fulfills these hierarchy management tasks.

'*A concept should not be declared directly or indirectly as subconcept of itself.*'

According to interpretation of *RelBAC*, we know that the antisymmetry property of the partial order '$\sqsubseteq$' applies in the hierarchies in groups, classes and permissions just as the subset-of relation between sets (a permission is also a set with pairs as elements). To enforce this constraint, we can refer to *subsumption* checking. Given two groups U_1, U_2, *RelBAC* checks the knowledge base

$$\mathcal{P} \models U_1 \sqsubseteq U_2?$$

A 'Yes' answer restricts '$U_2 \sqsubseteq U_1$' to be added to $\mathcal{P}$. For example, *a sales manager is also an employee* can be formalized as '$Manager \sqsubseteq Employee$'. If the administrator would like to assert by mistake that *an employee is a sales manager*, a pre-assertion checking of '$\mathcal{P}, \mathcal{S} \models Manager \sqsubseteq Employee?$' is processed and a 'Yes' answer avoids this operation which will create a 'loop' in group hierarchy.

'*A user should not be declared belonging to a concept and its subconcept at the same time.*'

According to set theory, given two groups U_1, U_2 and $U_2 \geq U_1$, for any user u, if $U_1(u)$ holds then we have $U_2(u)$ implied by the reasoner. Thus this constraint can be rephrased as '*a user should not be declared as member of a group which it already belongs to.*' We will discuss about the user membership management in Section 4.1.2 afterwards.

'*A set cannot be subset of two sets which are mutually exclusive.*'

This constraint holds because any set declared like that would be reasoned as empty set according to the following theorem.

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

For example, *no users are allowed to be assigned as members of both the sales manager and the sales agent* can be formalized as '$Manager \sqcap Agent \sqsubseteq \bot$'. Then any attempt to assign one user to both groups will be checked out as a conflicting operation. This could be used for separation of duties as in Section 4.1.4.

Here we talked about user group hierarchy only. For object class hierarchy and permission hierarchy, the theory applies similarly.

4.1.2 Membership Management

Users in an online community belong to various groups such as 'Friend', 'Soccer Fans', etc. *RelBAC* provides an access control mechanism based on membership of groups such as *users in the Friend group are allowed to read my music folder.* With the growing size and number of the groups, the management of user membership becomes crucial. *RelBAC* can help the administrator to control these memberships with reasoning. As object classes and permissions are only sets with different individuals (objects as individual of class, subject-object pair as individual of permission), the membership management of classes and permissions is similar to that of groups. Here in this section we discuss group membership management only and the theory applies on class and permission memberships.

Before talking about how $RelBAC$ reason on membership management, we define the entailment reasoning tasks of the $RelBAC$ reasoner coherent to classical Description Logic [4].

Definition 3. Entailment Given a $RelBAC$ knowledge base consists of $\mathcal{P}$ and $\mathcal{S}$, the $\mathcal{S}$ entails a piece of knowledge α with respect to $\mathcal{P}$ if the model for $\mathcal{P}$ and $\mathcal{S}$ is also a model for the knowledge. It is to check the following reasoning task.

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

Then from now on, *entailment* is regarded as a service provided by a DL reasoner that we can use as the output of a blackbox. This service can be also used in the following ways.

Consistency To check whether the knowledge base is consistent can be achieved by an entailment reasoning with $\alpha = \bot$. That is to query to the reasoner with

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

Redundancy If existing knowledge base already implies the assertion intended to be added, the assertion is redundant. A redundancy checking is an entailment as follows.

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

A 'Yes' answer from the reasoner means that this membership is not necessary because the knowledge base can infer this already.

Conflict If the knowledge base is consistent, but not consistent any more after adding the assertion, the assertion is conflicting with the knowledge base. A conflict checking is a consistency checking of the consistent knowledge base updated with the new assertion added.

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

A ‘No’ answer from the reasoner means that the updated knowledge base is still consistent and the membership of u to U_i can be added.

With these reasoning services provided by *RelBAC*, we can reason about the access control knowledge base for the membership management. The daily management of user membership includes mainly two parts: verification and update. Verification is to validate the membership of a given user to a given group. This can be achieved by the *entailment* service of the DL reasoner. For example, to verify *whether Hill is an employee* is an entailment reasoning as the following.

$$\mathcal{P}, \mathcal{S} \models \mathit{Employee}(\mathit{hill})?$$

The update of user membership means to add (delete) a user to (from) an existing user group which is in turn only to add (delete) an assertion to (from) the state base $\mathcal{S}$. For example, to add a user *hill* as a member to the *Employee* group, we can just add to $\mathcal{S}$ one assertion $\mathit{Employee}(\mathit{hill})$. This update of $\mathcal{S}$ will bring u all the permissions assigned to *Employee* just as the assignment of a role in RBAC.

However, before adding this state to $\mathcal{S}$, an administrator should check the *redundancy* and *conflict*. A simple algorithm of adding a subject to a given group is provided as Algorithm 3.

For example, to add the membership that *Hill is an employee* can be achieved with the following steps.

1. Redundancy checking with $\mathcal{P}, \mathcal{S} \models \mathit{Employee}(\mathit{hill})?$; if ‘Yes’, the assertion is redundant and not to be added to the knowledge base.

2. Conflict checking with $\mathcal{P}, \mathcal{S} \models \neg \mathit{Employee}(\mathit{hill})?$; if ‘Yes’, the knowledge base implies that *Hill is not an employee* and the assertion should not be added.

3. Update the knowledge base by adding $\mathit{Employee}(\mathit{hill})$.

Algorithm 3: Add Membership

```
Input: Subject u, Group G
if G(u) is NOT redundant for the knowledge base then
    if G(u) is NOT conflicting with the knowledge base then
        Add to the knowledge base with formula G(u);
        for each group $G_i$ in the knowledge base do
            if $G \sqsubseteq G_i$ then
                Remove the formula $G_i$(u) from the knowledge base;
            end
        end
    else
        Error(”Conflicting with existing knowledge!”);
    end
else
    Error(”Membership already assigned!”);
end
```

4. Delete any other redundant membership assignment for those groups that *hill* may inherit the membership from *Employee*.

The feature of *RelBAC*, the natural formalization of hierarchy, brings ‘free’ membership inheritance through user hierarchies. By ‘free’ we mean that no extra rule is necessary to specify the path of propagation after the hierarchy is clearly designed and expressed. For example, if *Hill is a powerful agent* and ‘$PowerfulAgent \geq Agent$’, then *Hill is a sales agent* comes for free with the following reasoning.

$$\{PowerfulAgent(hill), PowerfulAgent \sqsubseteq Agent\} \models Agent(hill)$$

That is to say, membership inheritance depends only on the group hierarchy. Given any two groups U_i, U_j such that $U_i \geq U_j$ and u is a member of U_i then the membership of u to U_j can be automatically implied by the reasoner. Notice the transitivity of partial order is preserved by subsumption '$\sqsubseteq$' in DL. In this case, it is not necessary that U_i, U_j are directly connected in the hierarchy.

When adding a user u as a member of a group U_i in a group hierarchy, we can do the same as above Algorithm 3, because the user hierarchy does not bring exceptions for the redundancy and conflict checking.

Here we have discussed the membership update of adding a user to a group. The other operation of the update management, to delete a user's membership from a group is just to remove an existing formula from the knowledge base if we have only isolated groups in the knowledge base, otherwise membership inheritance through group hierarchies should be considered.

To delete a user u from U_i, we have to consider those groups U_j that inherits the membership of u from U_i. In order to get the most specific group the user u belongs to, we introduce the *realization* task of $RelBAC$ coherent to classic Description Logic [4].

Definition 4. Realization Given an instance a and some concepts $C_1, ..., C_n$, find the *most specific concept* C_i in the given concept set such that $\mathcal{P}, \mathcal{S} \models C_i(a)$. Here the concepts can be a set of groups (or classes) and the reasoner will return the most specific group (or class) that a belongs.

As shown in Figure 4.1-a, the most complicated situation is that the membership of u to U_i is inherited from U_j which satisfies $U_j \geq U_i$ when some intermediate group U'_j exists. Thus the deletion of u from U_i requires the compensation of adding u as the member of all U_k, U'_k....

For example, an access control system for sales force automation [82] may have the group hierarchy as shown in Figure 4.1-b as a DAG. To delete the membership of a user 'Hill' from the group 'Employee', the algorithm finds the *realization* of *hill* is in the group of 'Powerful Agent'. By inheritance, *hill* has the membership to all the groups list in the figure.

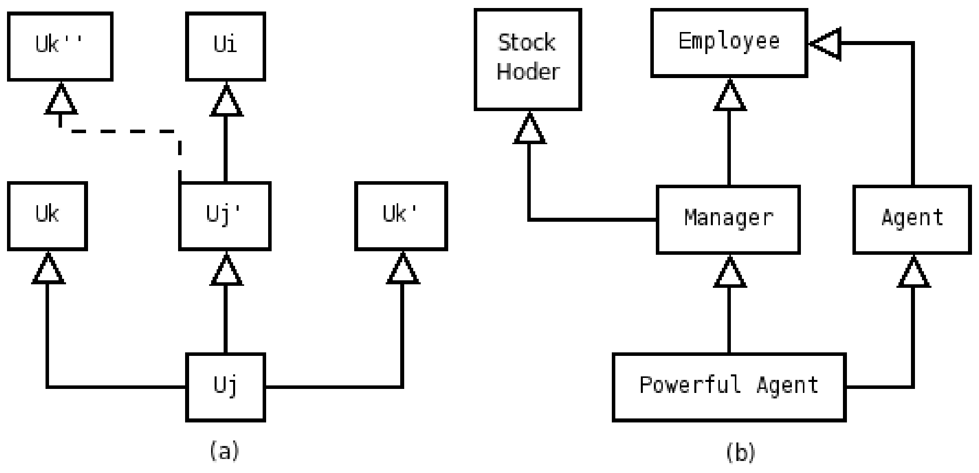


Figure 4.1: Delete User Membership in Hierarchy.

The membership delete operation should not affect the membership on group ‘Stock Holder’.

In Algorithm 4, ‘Powerful Agent’ is put into the list of ‘Most Specific Group’ in Line 3. The ‘for’ loop on Line 4 removes from the list of Most Specific Groups those groups that are not more specific than ‘Employee’. Then in the ‘for’ loop on Line 9, each group membership is removed by compensating the more general group with the membership which is ‘Manager’ and ‘Agent’. Because the membership to ‘Employee’ has not been deleted with this round (the ‘while’ loop on Line 2 remains true), the algorithm searches for new realization on Line 3 and finds ‘Manager’ and ‘Agent’. This round of compensation on Line 18 adds membership to ‘Stock Holder’ and ‘Employee’. It also removes the membership from ‘Manager’ and ‘Agent’. Yet another round in the ‘while’ loop, the membership from ‘Employee’ is removed and the algorithm terminates. The intuition of Algorithm 4 is to avoid the side-effect of the deletion operation such as to deprive some membership that is not intended.

Similarly, an object membership can be inherited through object hierarchies just as the subject membership inheritance. Therefore, the mem-

Algorithm 4: Delete a User Membership in Group Hierarchy

```
Input: Subject s, Group G
Group[ ] MostSpecificGroup;
while G(u) is entailed by the knowledge base do
    MostSpecificGroup = Realization(u);
    for each Group $G_i$ in MostSpecificGroup do
        if Not $G_i \sqsubseteq G$ then
            remove $G_i$ from MostSpecificGroup;
        end
    end
    for each Group $G_i$ in MostSpecificGroup do
        Group[ ] GTemp;
        for each Group $G_j$ in the knowledge base do
            if $G_i \sqsubseteq G_j$ then
                GTemp.add($G_j$);
            end
        end
        Delete the formula $G_i$(u) from the knowledge base;
        for each Group $G_k$ in GTemp do
            Add the membership of $G_k$ to u;  //Refer to Algorithm 3
        end
    end
end
```

bership management tasks for object can be achieved the same.

4.1.3 Permission Assignment

To assign to a user u certain permission P on an object o, $RelBAC$ adds an assertion $P(u, o)$ to the $\mathcal{S}$ in the knowledge base. Before this assignment operation, *redundancy* and *conflict* checking should be performed as for membership management in Section 4.1.2, to make sure that the permission has not been assigned or conflicting with existing assignments.

Sometimes the administrator tends to assign a group of users a permission on a class of objects. With rules (3.4-3.11) as shown in Section 3.2, $RelBAC$ provides easy ways to assign this with one rule rather than several rules for each possible (u, o) pair. Further more, this enables $RelBAC$ some other way as listed below to assign to a user u the permission P on an object o, where U_i is a user group and O_j is an object class.

1. Assign to u directly the permission P on the object o.
2. Suppose u is a member of a group U_i and the assignment of P on o to all the members of U_i will assign P on o to u.
3. For an object class O_j that o belongs to, to assign to u the permission P on all the objects of O_j will assign P on o to u.
4. If there is another permission P' more powerful than P, such that $P' \geq P$, the assignment to u the permission P' on o implies the assignment of permission P on o to u.

For example, Hill is a sales manager which is also an employee, and there is an offer named 'Trento'. The permission to update the case is more powerful than to read it. Then in order to specify that *Hill is allowed to read the offer named 'Trento'* in $RelBAC$, the following forms of rules can be added to the knowledge base.

1. $Read(hill, trento)$
2. $Employee \sqsubseteq Read : trento$
3. $(\forall Offer.Read)(hill)$
4. $Update(hill, trento)$

The first rule adds directly $P(u, o)$ as to assert *Hill is allowed to read the offer 'Trento'*. The other three assignments will also allow this permission with the help of reasoning, because each of the last three (together with the knowledge base) implies the first. To be exact, the second assignment grants access to all users in group U_i; the third grants u access to all objects in O_j; and the last assignment assigns to u some other permission P' which is more powerful than P.

Assignment 2 and 3 are relatively good features of *RelBAC* because it reduces the number of rules. By assigning to the largest group the permission on the largest class, the m (or n) rules in form of Assignment 1 are expressed with one single rule (m, n are cardinality of sets U_i, O_j respectively). Well, the fourth assignment should be considered twice before usage because it might lead to security leakage such as when some P'' is not intended to be assigned but propagated from P' with some background knowledge '$P' \sqsubseteq P''$'. For example, if the knowledge base contains that $\mathit{Update} \sqsubseteq \mathit{Read}$, $\mathit{Update} \sqsubseteq \mathit{Execute}$ and when we assign $\mathit{Update}(\mathit{hill}, \mathit{trento})$, not only *Read* is assigned to *hill*, but the permission *Execute* is implicitly assigned although not intended.

In any case, before assigning these rights, redundancy and conflict checking are necessary as for membership management in Section 4.1.2.

In contrast to membership inheritance, the permission propagation is more complex because a *RelBAC* permission is a binary relation that links a subject to an object. Therefore, it has three paths to propagate: user group hierarchy, object class hierarchy and permission hierarchy. Fortunately, these propagations come for free just as the output of automated reasoning and support tools are provided to help in the management work.

Rule 3.1 in Section 3.2 provides the way to build group hierarchy. For permissions assigned with *subject-oriented* rules, they will propagate from junior group to senior groups as

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

in which α stands for some permission together with the object to access. For example, $Manager \geq Employee$ implies that all the permissions assigned to the employees will propagate to sales managers. To be precise, if *each Employee is allowed to read minimum 10 offers* then the permission propagates to group sales manager such that *each sales manager is allowed to view minimum 10 offers* as is implied by the reasoner without any additional propagation rules.

$$\{Employee \sqsubseteq \geq 10 Read.Offer, Manager \sqsubseteq Employee\}$$
$$\models Manager \sqsubseteq \geq 10 Read.Offer$$

In addition to the group hierarchy which simulates the role hierarchy in RBAC model, *RelBAC* provides (object) class hierarchy and permission hierarchy with partial order '$\geq$' applied on classes and on permissions with respect to Rule 3.2 and Rule 3.3 in Section 3.2 .

Besides the subject hierarchy as paths for permission propagation, the object hierarchy also provides such paths. For two assignments β, β' of the same permission but on different object classes O_i, O_j, if $O_i \geq O_j$ the propagation goes different ways according to the semantics of the assignment.

- If $U \sqsubseteq \beta, U \sqsubseteq \beta'$ are assignments as Rule 3.4, 3.6 or 3.8 then

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

  because β assigns the permission onto *some (only, minimum n)* objects in O_i then it assigns on *some (only, minimum n)* objects in a superset of O_i, say O_j.

  For example, *the employees are allowed to read some (only, minimum 10) urgent offers* implies that *they are allowed to read some (only, minimum 10) offers* because urgent offers are a sub class of offers by the assertion $Urgent \sqsubseteq Offer$.

- If $U \sqsubseteq \beta, U \sqsubseteq \beta'$ are assignments as Rule 3.10 or 3.24 then

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

  because β' assigns the permission onto *all (maximum n)* objects in O_j, then it assigns the same permission on *all (maximum n)* objects in a subset of O_j, say O_i.

  For example, *the employees are allowed to read all (maximum 5) offers* implies that *they are allowed to read all (maximum 5, maybe less) urgent offers* because $Urgent \sqsubseteq Offer$.

Moreover, permissions can propagate through permission hierarchy as well. In contrast to sets of individuals such as groups or classes, the partial order among permissions describes *subsumption* between sets of (u, o) pairs. As shown in Rule 3.3 of Section 3.2 any individual pair (u, o) of P_i is also pair for P_j.

For example, $Update \sqsubseteq Read$ implies that any assignment with permission $Update$ is also assigned with $Read$ such as *the employees are allowed to update some offers* implies that *they are allowed to read some offers.*

The advantage of $RelBAC$ is that these propagations come for free when we define the hierarchies. No specific propagation rules are needed for the knowledge base. This feature will simplify the system design and reduce the possibility of errors.

4.1.4 Separation of Duties (SoD)

Advanced access control models tend to support separation of duties as an important security property. In this section, we first talk about SoD in general, then the dynamic SoD as a plus, last but not least, we discuss the adaptability of $RelBAC$ to support 'high-level' concerns [58] about SoD.

General SoD

This constraint states if a sensitive task consists of two steps, then a different user should perform each step. More generally, when a sensitive task is composed of n steps, a SoD constraint requires the cooperation of at least k $(2 \leq k \leq n)$ different users to complete the task.

The RBAC model [29] supports SoD as we have shown in Section 2.2.2 especially in the case when $k = 2$. However, n new RBAC roles should be used to represent the n steps of the SoD. This leads to the creation of new roles every time defining a new step. In *RelBAC*, a permission is a relation that links a subject with an object, and to enforce SoD the only thing to do is to assert directly the axioms about permissions without additional concepts such as RBAC roles to be created.

For example, *to create and to delete an offer should be separated in the SFA scenario* [82]. It can be expressed in *RelBAC* as

$$\exists Create.Offer \sqcap \exists Delete.Offer \sqsubseteq \bot$$

in which *Create* and *Delete* are two permissions. This policy restricts any pair (u, o) belongs to both *Create* and *Delete*.

In general, a SoD enforces that given n steps of a task, at lest $k(2 \leq k \leq n)$ users should be involved to fulfill the task. Suppose the worst case that users are of relatively equal ability to perform these n steps. Then, on average each user may take m steps of the task, where m satisfies the following equation.

$$\frac{n}{k} \leq m < \frac{n}{k-1}$$

Therefore, maximumly $m = \lceil n/(k-1) \rceil - 1$. To enforce the SoD, no user is able to fulfill any arbitrary $m + 1$ steps of the task. Then *RelBAC* can formalize this as follows.

$$\bigsqcup_{i=1}^{C_n^{\lceil n/(k-1)\rceil}} (\sqcap_{j=1}^{\lceil n/(k-1)\rceil} U_{ij}) \sqsubseteq \perp \tag{4.1}$$

in which U_{ij} stands for the (user) group that has any of the m rights. For example, suppose that to create, update, review and archive an offer are the 4 steps of processing an offer in the SFA scenario. A SoD requires that *minimum 4 faculty members should be involved in this process.* This SoD can be enforced as follows in $RelBAC$.

$$\begin{aligned}&(\exists Create.Offer \sqcap \exists Update.Offer) \sqcup (\exists Create.Offer \sqcap \exists Archive.Offer) \sqcup \\ &(\exists Create.Offer \sqcap \exists Review.Offer) \sqcup (\exists Update.Offer \sqcap \exists Review.Offer) \sqcup \\ &(\exists Update.Offer \sqcap \exists Archive.Offer) \sqcup (\exists Review.Offer \sqcap \exists Archive.Offer) \sqsubseteq \perp\end{aligned}$$

as $C_n^{\lceil n/(k-1)\rceil} = C_4^{\lceil 4/3\rceil} = C_4^2 = 6$. It intuitively enforces that any two steps cannot involve the same user.

Generally speaking, when $k = 2$, Rule 4.1 enforces a sufficient and necessary condition of the SoD. However, for cases $k \geq 3$, Rule 4.1 is still sufficient but not always necessary. In the example above, the rule we enforce for $k = 3$ is exactly the one we use for $k = 4$. It is understandable, that 'minimum 4 users' covers the case of 'minimum 3 users'.

In [59] N.Li et al. modeled the problem of $k - n$ static separation of duties with a first order logic formula as follows.

$$\forall u_1 ... u_{k-1} \in U((\bigcup_{i=1}^{k-1} auth\_perms_\gamma[u_i]) \not\supseteq \{p_1 ... p_n\}) \tag{4.2}$$

The formula 4.2 uses universal quantifier on arbitrary $k-1$ users in space of U. It collects all the permissions explicitly/implicitly assigned to this $k-1$ users and enforces that they are not superset of all the n steps (duties). Their solution has the complexity of $(|U|^{k-1} * n)$ which explodes to the cardinality of the subject space. Our solution enforces a sufficient but not always necessary condition of the SoD. However the complexity is only $(n^{n/k})$. Considering that the number of steps which is n, is usually far less than the number of users $|U|$ in the system, our method is more efficient than [59] in most cases.

Dynamic SoD

SoD is categorized into *static* and *dynamic* in RBAC[29] according to the time when the policy is enforced . An SoD enforced at design time is regarded as static, and as dynamic if it is enforced at run time. Static SoD can be represented in *RelBAC* as the general SoD discussed at the beginning of this section. Dynamic SoD should be enforced at run time, which intuitively means that the duties to be separated can be assigned but cannot be activated simultaneously at run time. RBAC fulfills it with the concept of session as representatives of the user at run time by restricting sessions from activation of separated roles. In *RelBAC*, dynamic SoD is achieved by *run time permissions.*

Definition 5. Run time Permission (*RTP*) A run time permission is a permission describing the execution of a common sense permission which has a name in the form of a verb (phrase). For each common sense permission, a corresponding *RTP* exists in the form of the present continuous tense of the verb.

For example, the *RTP* of permission 'Read' is 'Reading' and the *RTP* for permission 'ConnectOnWeekends' is 'ConnectingOnWeekends'. A user cannot activate a *RTP* unless she has the original permission which is enforced with the policy as *P-ing* $\sqsubseteq$ *P* when the *RTP* is introduced to *P*. For example, *Reading* $\geq$ *Read* is used to restrict that *a user cannot execute permission 'Reading' without the permission 'Read'.* Thus the dynamic

SoD '*a user cannot create and delete an offer at the same time*' is enforced as follows.

$$\exists Creating.Offer \sqcap \exists Deleting.Offer \sqsubseteq \bot$$

In the real world, a user can be grant the permission to create and to delete, but cannot be performed simultaneously.

To enforce dynamic SoD, the access control system should be informed of the state changing in time such that the real time state e.g., *Alice is creating an offer 'Bolzano'*. Then the knowledge base should be updated with the new assertion $Creating(alice, bolzano)$. Then the dynamic SoD will take effect that *Alice is not allowed to create and delete an offer at the same time* which is an entailment of the reasoner that

$$\mathcal{P}, \mathcal{S} \sqcup \{Creating(alice, bolzano)\} \models \neg\exists Deleting.Offer(alice)$$

although Alice might has both the permission to create and to delete some offer(s).

High Level Constraint of SoD

For the general SoD property, the composition of the k users to complete the task is sometimes important. The administrator may like to constraint first that these users are from certain groups, cardinality related constraints are also necessary for the composition. N.Li et al. studied SoD in detailed requirements for user attributes in addition to user number. An algebra is proposed in [58] to specify complex policies combining requirements on user attributes and number. On top of the number restriction for given duties, their algebra can specify the composition of the users for the SoD which they regard as *high-level* policy. For example, beyond the constraint that to process an offer should involve minimum 4 users, it can restrict that the set of users that can manage offers consists the following compositions.

1. Exactly 4 employees, i.e., 1 sales manager and 3 sales agents.

2. Minimum 4 employees including 1 sales manager and 3 sales agents and maybe more users as sales manager or agent.

3. Minimum 4 employees including 1 sales manager and 3 sales agents and maybe more employees.

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

$$\begin{aligned}
\mathit{Offer} \sqsubseteq & (= 4 \mathit{Process}^{-1}.\mathit{Employee}) \sqcap \\
& (= 1 \mathit{Process}^{-1}.\mathit{Manager}) \sqcap (= 3 \mathit{Process}^{-1}.\mathit{Agent}) \\
\mathit{Offer} \sqsubseteq & (\forall \mathit{Process}^{-1}(\mathit{Manager} \sqcup \mathit{Agent})) \sqcap \\
& (\geq 1 \mathit{Process}^{-1}.\mathit{Manager}) \sqcap (\geq 3 \mathit{Process}^{-1}.\mathit{Agent}) \\
\mathit{Offer} \sqsubseteq & (\geq 1 \mathit{Process}^{-1}.\mathit{Manager}) \sqcap (\geq 3 \mathit{Process}^{-1}.\mathit{Agent})
\end{aligned}$$

Here the permission *Process* can be regarded as the union of the 4 steps as $\mathit{Process} \equiv \mathit{Create} \sqcup \mathit{Update} \sqcup \mathit{Review} \sqcup \mathit{Archive}$. We abbreviate the usage of both $\geq n$ and $\leq n$ as $= n$ in *value restriction.* In addition to this kind of *high-level* policy in addition to the general SoD policy as discussed in Section 4.1.4.

4.2 Run time Reasoning

For a *RelBAC* based access control system, after the knowledge base is built with consistent policies and states (respect to $\mathcal{S}$ and $\mathcal{P}$), access control decisions can be made by reasoning about the access query and the knowledge base on an off-the-shelf reasoner.

Basically, to decide whether a user u has some permission P on some object o, Algorithm 1 is used. *RelBAC* submits a query for knowledge

$P(u, o)$ to the knowledge base predefined with $\mathcal{P}$ and $\mathcal{S}$. The decision is 'Yes' if $P(u, o)$ is in the knowledge base or implied by the knowledge base; otherwise, the decision is 'No'.

In addition to this basic decision on a query as $P(u, o)$, *RelBAC* is able to make decisions to more complex access requests. Given a group of users U which u belongs to, some permission P and a class of objects O which o belongs to, *RelBAC* provides access control decisions to the following queries.

1. Is the user u allowed to access the object o with the permission P? For example, *is an employee Hill allowed to read an offer 'Trento'?* can be a query as

$$\mathcal{P}, \mathcal{S} \models Read(hill, trento)?$$

2. Is the user u allowed to access some objects in O with P? For example, *is Hill allowed to read some offers?* can be a query as

$$\mathcal{P}, \mathcal{S} \models (\exists Read.Offer)(hill)?$$

3. Is the user u allowed to access maximum/minimum or equal to n of the objects in O with P? For example, *is Hill allowed to read maximum 5 of the offers?* can be a query as follows.

$$\mathcal{P}, \mathcal{S} \models (\leq 5Read.Offer)(hill)?$$

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

4. Is the user u allowed to access all the objects in O with P? For example, *is Hill allowed to read all the offers?* can be a query as follows.

$$\mathcal{P}, \mathcal{S} \models (\forall \mathit{Offer}.\mathit{Read})(\mathit{hill})$$

5. Is there any user(s) in U allowed to access all objects in O with permission P? For example, *is there any employees allowed to read all the offers?* can be a query as

$$\mathcal{P}, \mathcal{S} \models \mathit{Offer} \sqsubseteq \exists \mathit{Read}^{-1}.\mathit{Employee}?$$

because the virtual group implied by $\exists \mathit{Read}^{-1}.\mathit{Employee}$ is the set of all the objects that can be read by some employee.

6. Are there maximum/minimum or equal to n users in U allowed to access all the objects in O? For example, *is there minimum 3 employees allowed to read all the offers?* is a query as

$$\mathcal{P}, \mathcal{S} \models \mathit{Offer} \sqsubseteq \geq 3\mathit{Read}^{-1}.\mathit{Employee}?$$

7. Is each user of U allowed to access maximum/minimum or equal to n objects in O? For example, *is each employee allowed to read more than 10 offers?* can be a query as follows.

$$\mathcal{P}, \mathcal{S} \models \mathit{Employee} \sqsubseteq \geq 11\mathit{Read}.\mathit{Offer}?$$

8. Is each user of U allowed to access all objects in O with P? For example, *is each of the employee allowed to read all the offers?* can be a query as

$$\mathcal{P}, \mathcal{S} \models \mathit{Employee} \sqsubseteq \forall \mathit{Offer}.\mathit{Read}?$$

because the virtual class implied by $\forall \mathit{Offer}.\mathit{Read}$ is a set of all the users that can read all the internal grades.

We can see from the above that quite flexible queries can be answered by the reasoner such that complex access control can be decided e.g., cardinality related queries. The expressiveness of *RelBAC* model allows to have 15 types of rules for fine-grained access control as discussed in Section 3.7.

4.3 Summary

RelBAC allows to express many complex properties, and especially powerful in cardinality related policies. In this chapter, we discussed the automated reasoning issues about *RelBAC*. Management tasks on hierarchy, membership, and permission assignment can be translated into design time reasoning for *RelBAC*. Important security properties such as Separation of Duties (SoD) are also discussed here with an even further step on the high-level SoD which concerns the composition of the users sharing the duties. Access control decision can be made on various access requests according to the run time reasoning of *RelBAC*.

In the next chapter, we will show the extensibility of *RelBAC* by discussing how to capture the features of RBAC model and its extensions, e.g., PBAC model.

Chapter 5

Other Models in *RelBAC*

RelBAC is a flexible access control model with great extensibility. It focuses on the basic access control components only, so additional components can be easily extended without conflicts; it models the permission as a binary relation connecting the subject and the object, so that the complex policies can be naturally addressed; it is built on an Entity Relationship model so that it can be integrated into a large system design to cooperate with other parts; and it is formalized with a Description Logic based logical framework so that any unary and binary relations can be easily captured and extended.

We are going to show in this chapter how other access control models are represented in *RelBAC*. Section 5.1 discusses the early access control models such as AM and ACL; the extension to RBAC model is addressed in Section 5.2; in Section 5.3 we represent PBAC model in *RelBAC* and summarize the extensibility of *RelBAC* in Section 5.4.

5.1 Early Models in *RelBAC*

As discussed in Chapter 2, early access control models evolve from the original basic access control to AM[10], and then to ACL[76]. Here in this section, we show how *RelBAC* represents these early models.

As shown in Chapter 3, $RelBAC$ has the same entity sets as the basic access control model in Figure 2.4, which are *Subject, Object* and *Operation*. The difference is that the *Permission* in $RelBAC$ plays two roles. One role as the operation is by naming a *permission* with exactly the name of the *Operation*; the other role as the *Policy* relation among the other three is by connecting the *Subject* and the *Object* as a binary relation.

Access Control Matrix (AM) and Access Control List (ACL) models express the same intention in different representation of the basic access control model. An AM shows the 3-ary relation with a 2-dimensional matrix in which the contents of the cells represents the *Operation*. An ACL is attached to an *Object* as a list composed of (*Subject, Operation*) pairs. Thus, when a request comes for the object, the system checks if the subject and the intended operation are in the list.

$RelBAC$ can be regarded as to represent the cell of an AM model with a named pair

$$Permission(Subject, Object)$$

and avoids the drawback of a large matrix with large portion of empty cells. A named pair can express the same as an entry in an ACL does, and even more (in the sense that a named pair has the third parameter rather than the only two in an entry). $RelBAC$ provides more flexibility than ACL. For example, an ACL is attached to an object and it is hard to check from a subject point of view such as to query for all the documents that a given user can read. $RelBAC$ does not have such limits because it offers both *subject-oriented* and *object-oriented* rules in addition to the ground rules, exactly in the form of a named pair that can be queried from any perspective, e.g., given a subject and query, then query for the permissions and objects, etc.

Therefore, $RelBAC$ model captures all the features of the early access control models.

5.2 RBAC in *RelBAC*

As a prevailing access control model, RBAC has many advantages in expressiveness and efficiency. In this section we first present the interpretation of the RBAC model as basic access control policies. Then, we show how RBAC can be covered by *RelBAC*.

5.2.1 Interpretation of RBAC

As shown in Figure 2.3, the core of RBAC [29] is the *role*, which originates from social positions such as *manager* in an enterprise solution. Predefined roles simplify the assignment procedure into user-role membership management. Thus, subject and object are no longer directly related as in the basic access control model.

From the Figure 2.3 we can see that RBAC consists of additional components to the basic access control model in Figure 2.4 are in three aspects:

1. ROLES (hierarchy) intermediate USERS and PRMS (permissions[1]);
2. Combination of OPS (operations) and OBS (objects) into PRMS;
3. SESSIONS for run time constraints.

The first two differences split the permission assignment procedure into three steps: first, combine an operation p and an object o as a permission; then, assign the permission to a role r; last, assign a user u to the role r. It is exactly to make p, o a pair, and then connect the pair (p, o) to a role r which forms a tuple (r, p, o), and then form another pair (u, r) and by the role r shared in both tuples, u get the access to o with p. The advantage is that (p, o) pairs can be predefined and connected to r at design time. At run time, when a new user u comes, assignments of all (p, o) pairs are

[1]A pair as (operation, object), to be differentiated from a *RelBAC* permission.

simplified into one assignment of the user u to a role r which has been assigned those intended p, o pairs already. The efficiency is achieved as linear time to the number of users that need the assignments. A drawback of RBAC is that the pre-combination of p, o binds o to p such that no policy can be specified from the object point of view, e.g., to enforce that 'internal information can be accessed only by those people whose security level is above manager'.

SoD is an important constraint RBAC can enforce. In [29] D.Ferraiolo, R.Sandhu et al. distinguished static SoD with dynamic SoD according to the time this constraint is enforced. Static separation of duties (SSD) enforces constraints on the assignment of users to roles. Membership in one role may prevent the user from membership of one or more other roles. With the role hierarchy, inherited roles should also be considered for this constraint. Dynamic separation of duties (DSD) differs from SSD by placing constraints on the roles that can be activated within or across a user's sessions.

5.2.2 RBAC in *RelBAC*

The RBAC model has the following unary components which we can be represented in *RelBAC* as concepts: *ROLES, OPERATIONS, OBJECTS.* Binary components which can be represented as *RelBAC* roles[2]: *PERMISSIONS.* Details of the correspondence from RBAC to *RelBAC* are shown in Table 5.1.

We see that RBAC roles can be represented with the user groups in *RelBAC* and the user assignment (UA) in RBAC is just the user membership in *RelBAC*. RBAC permissions are (operation, object) pairs which do not exist in *RelBAC* because objects in a *RelBAC* model are decoupled from operation. However, a RBAC permission can be captured with the

[2] To be differentiated with a RBAC role, here a *RelBAC* role is a binary predicate in DL.

Table 5.1: RBAC in *RelBAC*

RBAC Component	***RelBAC* Component**
$USERS$	$SUBJECT$ instances
$ROLES$	$SUBJECT$ sets
$OPERATIONS$	$PERMISSION$ names
$OBJECTS$	$OBJECT$ instances
$SESSIONS$	encoded in *Run time PERMISSION*
UA	$SUBJECT$ membership
PA	*RelBAC* rules
RH	Subsumption axioms with '$\sqsubseteq$'

following formulas.

$$Op : ob \tag{5.1}$$

$$\forall Ob.Op \tag{5.2}$$

in which Op is a *RelBAC* permission with the name as the RBAC operation; ob is the object instance when the RBAC permission relates to an object; Ob is the object set when the RBAC permission relates to a set of objects. The intuition of Formula 5.1 and 5.2 is to represent a virtual group of users that all the group members can perform the operation Op on the object ob or on all the objects in Ob. The permission assignment (PA) can be achieved as the *RelBAC* Rule 3.12 and 3.24.

$$Role \sqsubseteq Op : ob \tag{5.3}$$

$$Role \sqsubseteq \forall Ob.Op \tag{5.4}$$

Role hierarchy in RBAC can be achieved with the partial order in *RelBAC* with Rule 3.3.

$$Role_1 \sqsubseteq Role_2 \tag{5.5}$$

Figure 5.1 illustrates the relation of RBAC with *RelBAC* by simulating

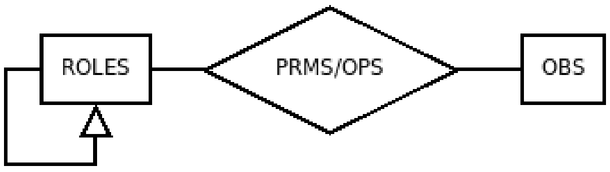


Figure 5.1: RBAC in *RelBAC*

the structure of *RelBAC* with RBAC terminologies.

RelBAC model can capture the three steps of assignment in RBAC as:

1. Define empty groups $G'_1, ...$ as roles in RBAC;

2. Connect these empty groups with object o with operation p;

3. Connect real user groups $G_1, ...$ (or user u) to $G'_1, ...$

For constraints such as SoD, we can take use of the predefined empty group 'G'_i...' and separate the duties by means of mutually exclusions. For example, a SoD takes form (rs, n) where $rs = \{G'_1, G'_2, G'_3\}$ and $n = 2$ then it can be formalized as axiom:

$$G'_1 \sqcap G'_2 \sqcup G'_1 \sqcap G'_3 \sqcup G'_2 \sqcap G'_3 \sqsubseteq \perp$$

One of the main features of RBAC is 'permission inheritance' (i.e. a child role inherits all the permissions assigned to its parent role). *RelBAC* uses the partial order 'generalization' for group hierarchy to simulate role hierarchy in RBAC and extends the hierarchy into objects and permissions.

In addition, we have many other kinds of permission assignments , which can easily assign operation P_j onto *some/only* objects in type T_k. The assignment onto *all* objects in T_k, which is the typical permission in RBAC, can be expressed with Theorem 1 with help of constructor *value restriction, complement of concept and role.*

The advantages of RBAC model are in three aspects:

1. Assignment of permissions is linear to user number;

2. Inheritance of permissions through role hierarchy;
3. Constraint as SoD can be enforced.

All of these features are preserved by *RelBAC*.

5.3 PBAC in *RelBAC*

Purpose Based Access Control (PBAC) [15] was proposed by J.Byun, E.Bertino and N.Li as an extension of RBAC with definition of 'purpose' in 2005. It addresses the necessity to control the access on the purpose of usage on the object with conditional roles. We will show PBAC in an ER diagram in this section, then interpret PBAC and at the end of this section we show how *RelBAC* captures PBAC.

5.3.1 PBAC model

As an extension to classic RBAC, the PBAC model consists of the following components:

PURPOSES access reasons, P as a set of purposes.

PURPOSE TREE a set of purposes organized as a tree structure denoted by PT, $Ancestors$ of P in tree PT is the set of all nodes that are ancestors of nodes in P, $Descendants$ of P is the set of all descendants of nodes in P and nodes in P;

ATTRIBUTES there are attributes for role and system description, and each attribute a_i has a data type τ_i.

CONDITIONAL ROLES Tuples as (r, C), where r is a role and C is a set of propositional logic formula constructed from primitive constraints with logical operator '$\vee(OR), \wedge(AND), \neg(NOT)$' in which primitive constraints are composed of attributes and/or purposes;

AUTHORIZATION Three-step procedure as in RBAC with minor fixes:

1. assign permissions to conditional roles;
2. assign access purpose to conditional roles;
3. assign user to conditional roles;

VERIFICATION Given an access request as a tuple (u, ap, p, o) respectively for *user, access purpose, operation and object*, the request is accepted only if all of the following are satisfied:

1. there exists a conditional role $cr = (r, C)$ where C is satisfied and permission (p, o) is assigned to r or inherited to r;
2. ap is assigned to cr or inherited to cr;
3. u is assigned to cr or inherited to cr;

Intended Purpose (IP) tuples are defined with the purpose tree as (AIP, PIP) where $AIP \subseteq P$ and $PIP \subseteq P$ are the sets of allowed and prohibited intended purposes respectively. The set of purposes implied by IP, denoted by IP^* is defined as

$$(Ancestors(AIP) - Ancestors(PIP) \cup Descendants(PIP))$$

Given the intended purpose IP over purpose tree PT, an access purpose ap is compliant to IP if and only if $ap \in IP^*$.

5.3.2 Interpretation of PBAC

The aim of PBAC is to solve the problem that the predefined roles do not adequately specify the set of users we wish to grant the access. J.Byun et al. use *conditional roles* to specify different attributes of the role and the system such that when a user activates the role, she must fill in the corresponding attributes with the values about the user. This allows the administrator to design different settings of the same role for different users. In contrast to the RBAC model, the additional information of purpose acts as a new parameter of both the role and the user. The original relation which is a triple (u, p, o) becomes a four-tuple as (u, ap, p, o) where u, p, o stand for the original *user, operation, object* respectively and ap stands for *access purpose*.

Moreover, PBAC differentiates from RBAC with the attributes and predicates over these attributes. In [15], attributes are divided into role attributes and system attributes. Role attributes specify characters of the users assigned to the role and make users distinguishable even within a role. Regarded as a parameter for the permission assignment, system attributes describe the state of a system and influence the definition of conditional roles. According to the state of the system (different values of the system attribute), many conditional roles (r, c_i) can be formed based on a single role r. In this perspective, PBAC just increases the number of roles in a flexible way that different conditions can be used. As the domains of an attribute may vary from 'time' to 'number' or even 'currency'... there would not be a common framework that can cover all these domains. Therefore, attributes should be defined according to system requirements respectively. But once the domain is fixed, corresponding predicates can be defined definitely.

PBAC is an example of many RBAC extensions that enrich the RBAC model with more parameters.

5.3.3 PBAC in *RelBAC*

From the interpretation, we see that PBAC offers more parameters for more precise permission assignments than RBAC.

With the ER diagram of Figure 5.2, we can see how PBAC is captured by *RelBAC*.

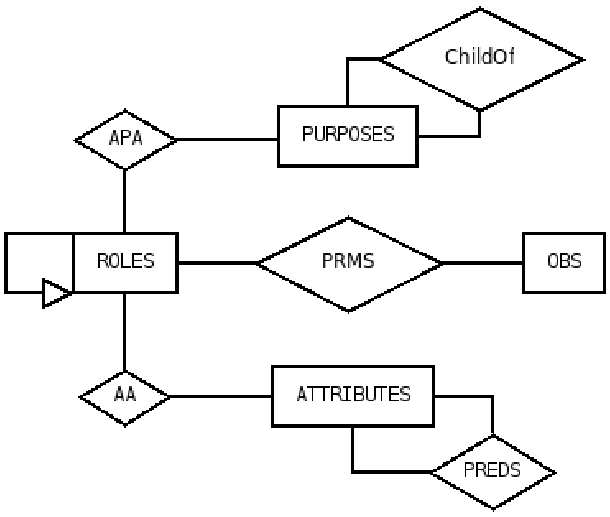


Figure 5.2: PBAC in *RelBAC*

An entity set named PURPOSES is added as the set of purposes to capture the component of purpose in PBAC. A 'child-of' relation can be used to form PURPOSE TREEs. A purpose is assigned to ROLES with 'access purpose authorization (APA)'.

Another entity set ATTRIBUTES is used to for the set of attributes. We use the relation 'Attribute Assignment (AA)' for 'role-attribute' assignments in PBAC. The PREDS relation stands for the predicates defined on attributes.

All these additional components can be easily supported in *RelBAC* with small extensions and formalized in DL framework, as shown in Table 5.2.

PURPOSES can be formalized with a concept as AP, a purpose such as 'direct' or 'market' is an instance of the concept; ATTRIBUTES can be formalized with concepts as $AT_1, ..., AT_m$; PREDS as the predicates between attributes can be formalized with roles as $PRED_1, ..., PRED_n$; access purpose can be assigned with an access control policy statement with a role $HasAP$ as

$$G_l \sqsubseteq \exists HasAP.AP;$$

Table 5.2: PBAC in *RelBAC*

PBAC Component	*RelBAC* **Component**
PURPOSES	AP: concept for all purposes;
ATTRIBUTES	$AT_1, ..., AT_m$: concepts for each attribute;
PREDS	$PRED_1, ..., PRED_n$: roles for predicates between attributes,
	$AT_i \sqsubseteq \exists PRED_j.AT_k$ attribute relation;
AA	$HasAt$: role for attribute assignment,
	$G_l \sqsubseteq \exists HasAt.AT_i$ group attribute assignment;
APA	$HasAP$: role for access purpose assignment,
	$G_l \sqsubseteq \exists HasAP.AP$ access purpose assignment;

group attribute can be assigned with a policy with role $HasAT$

$$G_l \sqsubseteq \exists HasAt.AT_i;$$

attribute relation can be formalized with a policy as

$$AT_i \sqsubseteq \exists PRED_j.AT_k;$$

Here in the formalism, i, j, k, l, m, n are all natural numbers in which m, n are the number of predicates and attributes respectively.

We can see from the *RelBAC* extension that all extra parts of PBAC model are covered. With the flexible expressiveness based on ER model and DL, *RelBAC* is powerful in extensibility.

5.4 Summary

In this chapter, we have shown the extensibility of *RelBAC*. At the beginning of the chapter, we have represented the early access control models in *RelBAC*. Then, with the popular RBAC model as an example, we have also shown that *RelBAC* can represent state of the art access control models without any difficulty. *RelBAC* offers even more expressiveness e.g., with object-oriented rules, *RelBAC* can enforce constraints from the object side; with the value restriction axioms, *RelBAC* can enforce the so-called high-level SoD. Further more, we take the PBAC model as a representative of RBAC extensions with more parameters such as purpose and attributes. *RelBAC* is easily extended to capture PBAC features. To support the predicates over attributes, we model each attribute as a concept with attribute value as the instance and model the predicates upon attributes as roles. As an advantage, DL framework brings *RelBAC* great expressiveness to model any unary and binary predicates.

Chapter 6

Lightweight Ontologies for Access Control

RelBAC is an access control model born for the web and it has close relation to the OWL [69], the web language for ontology. The logical framework of *RelBAC* is Description Logic, and the knowledge base can be expressed with OWL-DL language [71] into an ontology. The ontology, in turn can be put to a reasoner though the OWL-API [70]. We will show in this chapter the way to apply the theory and algorithms of Lightweight Ontologies [42] on *RelBAC*.

The chapter is organized as follows: Section 6.1 gives the permission propagation problem as a motivation; Section 6.2 briefly introduces the theory and algorithms Lightweight Ontologies; Section 6.3 shows how to apply the theory on *RelBAC*; and we conclude in Section 6.4.

6.1 Propagation Problem

Access right propagation is one of the most important features of *RelBAC*. The natural formalization of the partial orders inside the subjects, the objects and the permissions are free paths for the propagation. However, there are far more other relations rather than 'generalization'.

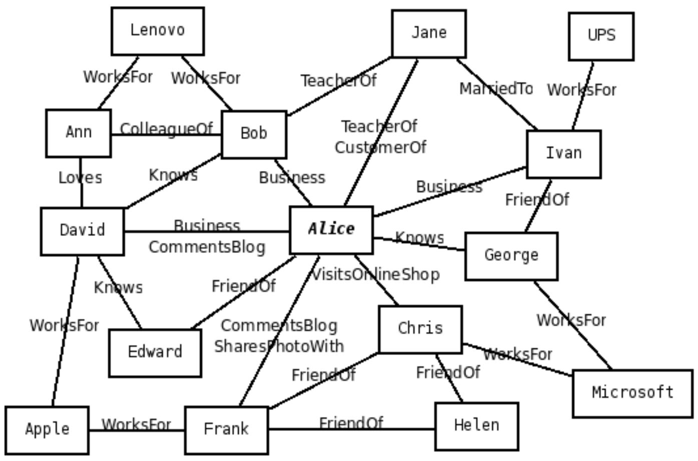


Figure 6.1: Alice's Social Network

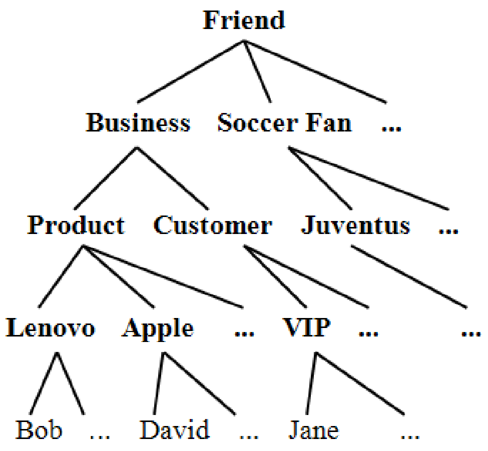


Figure 6.2: Alice's Social Ontology

We use an example of social network to illustrate the heterogeneity of relations among users. Suppose that Alice, an eBusiness vendor, has an online shop on eBay selling digital devices. Figure 6.1 shows part of her social network. For instance Bob, David and Ivan have business relations with her, while Chris and George are just common friends. Although the 6-degree separation [5] theory applies on the social networks nowadays, no constraints are placed on the type of relations between the 6 people. Foaf [67] and Web Of Trust [68] are typical social network relations, but there are thousands of other relations, e.g., comment on someone's blog, share the same research topic with, etc. Back to our example, with the continuous growth of her business, Alice decides to manage these contacts in her own way, so that she can easily find the 'proper' profiles whenever necessary. David is a 'business friend' who works at the sales department of Apple, and he will inform Alice about products and special offers such that Alice can immediately put them on her website. Jane is her 'best customer': she visits Alice's online shop frequently and comments on the deals she has just completed. This will help new potential customers in getting an impression of the service and quality of the goods. Of course, Alice is happy to give Jane VIP prices as rewards. Finally, Alice has a tree-like structure such as in Figure 6.2.

Notice that although people tend to organize his/her social network in a tree-like hierarchy, the relations between the nodes of this tree are not unified. Moreover, the social network is made of human beings that can easily move in or out of the network. Therefore, we come to the problem: how can we propagate the access right in this messy evolving network?

6.2 Lightweight Ontologies

A lightweight ontology originates in [35] for classification of objects, and now it is formally defined in [42] as a term for ontology in the field of semantic web.

Definition 6. A lightweight ontology is a triple of $\langle N, E, C\rangle$ where N and E are sets for nodes and edges that form a rooted tree, and C is a finite set of concepts expressed in a formal language F as the one and only one label for each node and if node n_i is the parent of node n_j the concepts satisfy that $c_j \sqsubseteq c_i$.

A propositional Description Logic language, i.e., a DL language without roles, is used as the formal language F. To build a lightweight ontology out of a natural language terms labeled tree, we follow the following steps.

1. The label of each node is transformed into a propositional DL formula using natural language processing (NLP) techniques. For example, the label 'Soccer Fan' is transformed into '$Soccer^i \sqcap Fan^j$' where the superscript $i(j)$ stands for the ith (jth) meaning of the word in a reference dictionary (e.g., WordNet).

2. Each node is associated a formula, called the *concept at node*, which is the conjunction of the formulas of all the nodes on the path from

the root to the node itself. For example, the node labeled 'Soccer Fan' in Figure 6.2 will be labeled with '$Friend^k \sqcap Soccer^i \sqcap Fan^j$'. The concept at node univocally defines the 'meaning of that node', namely, the set of documents which can be classified under it.

These two steps disambiguate the meanings of each natural language item and calculate the concepts for each node in order to meet the request that if node n_i is the parent of n_j then the concepts of n_i and n_j satisfy $c_j \sqsubseteq c_i$. Lightweight ontologies are not only a way to organize objects, but can be used on people such as to classify the acquaintances of Alice in Figure 6.1 into a lightweight ontology shown as Figure 6.2.

6.3 Lightweight Ontologies in *RelBAC*

As we discussed in Section 6.1, each person will belong to one or more different social communities where many people and different social relations will co-exist. With the communication simplified by the development of Internet, social activities such as online forums and blogs have greatly enriched the type of relations in a social network: not only traditional relations like 'knows', 'is-a-friend-of' etc. but new terms such as 'shares-photo-with' or 'comments-on-blog'. It is almost impossible to propagate permissions among these relations.

Moreover, people cannot find what they need in a messy web like Figure 6.1, but are familiar with tree-like structures such as the file systems of their computers, their email directory, classifications, catalogs, and so on. In general, there is a widespread tendency towards organizing resources (even the social community is a kind of resource) in tree-like structures. The key feature underlying the success of directories is that one can easily find something according to the property that, the deeper a category in a tree, the more specific resources it will contain. F.Giunchiglia et al.

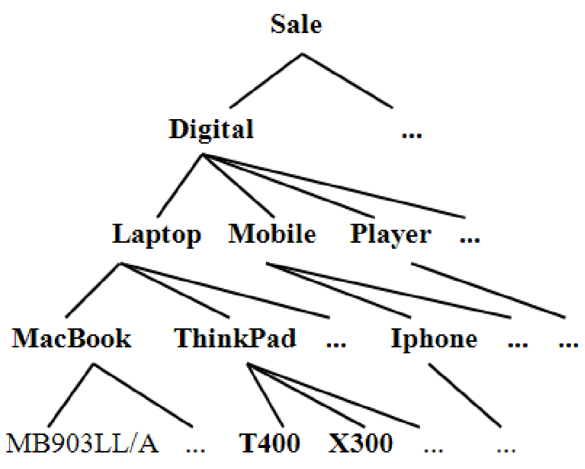


Figure 6.3: Alice's Web Directories

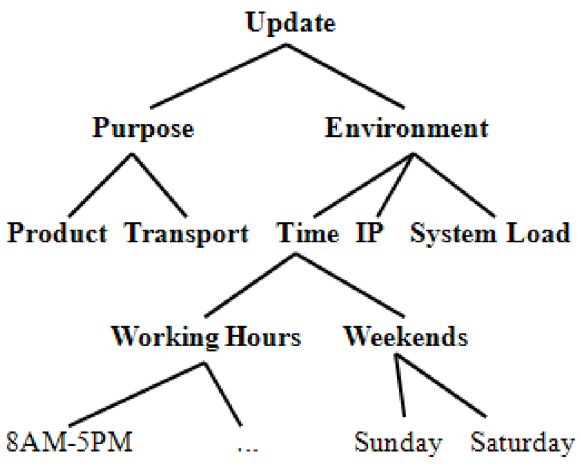


Figure 6.4: 'Update' Ontology

describe the details in [43] about the algorithms and implementations.

As a consequence, access control within user communities can be implemented in *RelBAC* as follows:

1. We implement subject hierarchies, as defined Section 3.2, and like the one in Figure 6.2, as lightweight ontologies of subjects (users, agents, customers, friends, ...);

2. We implement object hierarchies, as defined Section 3.2, and like the one in Figure 6.3, as lightweight ontologies of objects (resources, files, photos, web services, ...);

3. We implement permissions as relations and formalize them as DL roles between subject and object hierarchies. In turn, permissions as well can be organized as lightweight ontologies, like the one in Figure 6.4.

Notice that hierarchies like the ones in Figures 6.2, 6.3 and 6.4 allow to implement the partial order defined in Section 3.2. But not quite: to achieve this, we need to have DL formulas (and not natural language sentences) in nodes. Given that end-users most often do not understand and do not know how to manage logical formulas, our solution is to leave natural

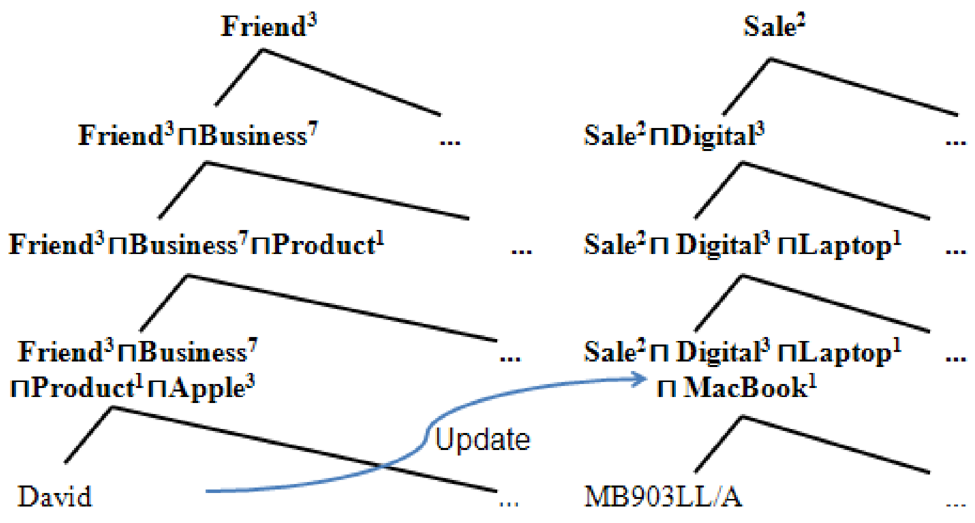

Friend³
Friend³⊓Business⁷
Friend³⊓Business⁷⊓Product¹
Friend³⊓Business⁷ ⊓Product¹⊓Apple³
David
Update
Sale²
Sale²⊓Digital³
Sale²⊓ Digital³ ⊓Laptop¹
Sale²⊓ Digital³ ⊓Laptop¹ ⊓ MacBook¹
MB903LL/A
...

Figure 6.5: *RelBAC* Permission Assignment on Lightweight Ontologies

language sentences on the user interface and to translate them, with no or very little user intervention, into DL formulas, thus obtaining lightweight ontologies with the same structure as the input hierarchy. We achieve this by exploiting the ideas described in [36, 37]. This translation happens according to the steps introduced in Section 6.2.

The user will keep seeing and managing hierarchies but all her operations will be supported and (partially) automated via reasoning on the underlying lightweight ontology. The IS-A hierarchy of this ontology makes precise and explicit all the user defined implicit and ambiguous relations between object categories. This substantially facilitates the access control problem. More concretely, some advantages are:

- Objects can be automatically classified into the proper directories with the help of a DL reasoner. With the algorithms described in [36, 37], it becomes possible to easily add the vast amount of new information to the proper categories with the proper access rules;

- The evolution of the object ontology (e.g., addition or deletion of a new category) is much more under control because it must satisfy the

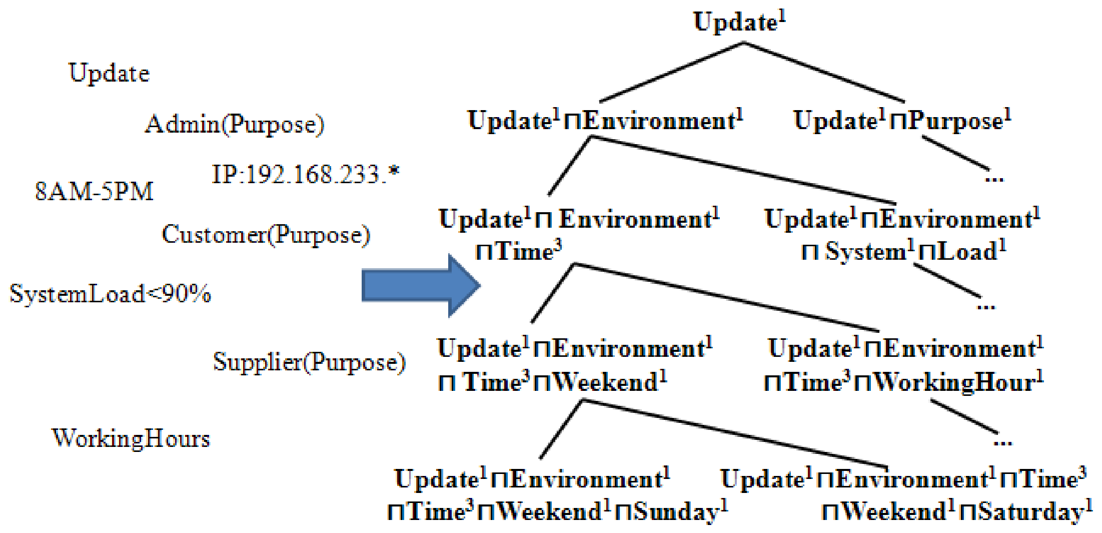

Update
Admin(Purpose)
IP:192.168.233.*
8AM-5PM
Customer(Purpose)
SystemLoad<90%
Supplier(Purpose)
WorkingHours
Update¹
Update¹⊓Environment¹
Update¹⊓Purpose¹
...
Update¹⊓ Environment¹ ⊓Time³
Update¹⊓Environment¹ ⊓ System¹⊓Load¹
...
Update¹⊓Environment¹ ⊓ Time³⊓Weekend¹
Update¹⊓Environment¹ ⊓Time³⊓WorkingHour¹
...
Update¹⊓Environment¹ ⊓Time³⊓Weekend¹⊓Sunday¹
Update¹⊓Environment¹⊓Time³ ⊓Weekend¹⊓Saturday¹

Figure 6.6: Scattered Permissions to a lightweight ontology

underlying ontological semantics;

- As from Section 3.2, the permissions on an object category will propagate up the tree.

We show in [88] the details on how reasoning can be exploited to achieve the above advantages, and more. Considerations similar to those provided for object ontologies apply also to subject ontologies. As mentioned above, these ontologies can be used to organize access to the underlying (possibly very messy) social network. There are however two further important considerations. The first is that *RelBAC* subject lightweight ontologies closely resemble the RBAC role hierarchies [29]. They are even easier to manage as users and permissions are decoupled. The second is that the links across subjects in a social network, like those in Figure 6.1, can be used to suggest candidate paths for permission propagation. One such small example is depicted in Figure 6.5. Finally, the translation into a lightweight ontology can be applied also to permission hierarchies. The result of applying this translation process to the hierarchy in Figure 6.4 is (partially) depicted in Figure 6.6. The natural language terms on the left of Figure 6.6 are meant to provide evidence of how the step from natural language to logic allows us to organize otherwise sparse categories into a lightweight ontology.

6.4 Summary

In this chapter we have shown how, based on the *RelBAC* formalization of the access control problem in social networks, we can organize users, objects and permissions as (lightweight) ontologies. The lightweight ontology theory and algorithms provide a way of managing the social community knowledge into tree-like structures and model permission assignments as binary relations between subject and object ontologies. This allows to represent access control rules and policies as DL formulas and therefore to reason about them using state of the art off-the-shelf reasoners. The major part of this chapter will be published in [?].

Chapter 7

Semantic Matching for Access Control

RelBAC has been proposed in [44] as a new model for the dynamic evolving community access control scenario. OWL-DL [71] can be used to represent the model as an ontology. As discussed in Chapter 6, Lightweight Ontologies [?] theory and algorithms provide a way to organize the subject, object and permissions into ontologies. This brings us not only the expressiveness, but also the possibility to apply modern semantic web techniques such as semantic matching [41] on these ontologies.

From an administration point of view, *RelBAC* provides for dynamic community access control with an adjustable, expressive and dis-ambiguous system specification and also powerful management and administration on information in various scales. However, to generate new rules on the fly for this vast amount of changes will be time-consuming and error-prone. Therefore, automated generation of access control rules becomes an interesting topic of research.

We present in this chapter a new way of applying semantic matching techniques on rule design and reuse in the access control domain. With a running example, we show how this proposal works and how the matching results are used.

The chapter is organized as follows: Section 7.1 shows the motivation of the request for new rules generation in a system based on *RBAC*; Section 7.2 introduces briefly the semantic matching techniques; Section 7.3 shows details of how to apply semantic matching on access control (lightweight) ontologies; and we summarize in Section 7.4.

7.1 Request for New Rules

RelBAC is a model for community access control. It has common components such as *SUBJECT, OBJECT* and a special part *PERMISSION* as binary relations. A domain specific (access control) Description Logic is used to formalize the *RelBAC* model. *SUBJECTs, OBJECTs* are formalized as concepts, and *PERMISSIONs* as roles. Hierarchies in the model can be formalized as subsumption axioms. All the system information and access control rules are logical formulas that can form a knowledge base as an ontology on which automated reasoning can be performed.

In the eBusiness scenario of Chapter 6, Alice may build a lightweight ontology as described in [36, 35, 42, 43]. To capture the various relations between persons throughout her social network and to control the access upon all these users in the network are simplified as to manage the links between subject and object ontologies. For instance, Figure 6.5 shows parts of the lightweight ontologies built by Alice and the assignment of 'Update' to user 'David' on the set of objects 'MacBook'. In the left lightweight ontology, David is classified as an instance of the set '$Friend^3 \sqcap Business^7 \sqcap Product^1 \sqcap Apple^3$' according to his social position that he has a $Business^7$ relation with Alice and he works for $Apple^3$ (the superscript depicts the 3rd sense in an external knowledge base which is an IT company rather than a fruit). In the right lightweight ontology, there's a class of objects '$Sale^1 \sqcap Digital^3 \sqcap Laptop^1 \sqcap MacBook^1$' where $Sale^1$ is a branch of $Business^7$, $MacBook^1$ is a $Laptop^1$ as a $product^1$ of $Apple^3$. Apparently the two concepts are different in labels, but semantically overlapping.

Things become more complicated when new ontologies arrive e.g., when Alice likes to collaborate with some other eBusiness vendors each of which has her own knowledge base of users and products (formalized as lightweight ontologies). A classical way to control the access on heterogeneous knowledge bases is to merge the knowledge and create new rules for the 'new' knowledge base. For example, Bob, another eBusiness vendor has his own social ontology as Alice. The collaboration of Alice and Bob requires integration of their social 'resource's, such as product supplier, transporter, customer, etc. in addition to the integration of the physical resources such as goods.

So the motivation lies in at least two phases, design and reuse.

- Semantic similarity may disclose some latent relationships between subjects and objects which appear 'irrelevant'. These latent relationships might suggest new rules that should be created for these semantically relevant subjects and objects such as to permit David to update the web categories about Apple products.
- Semantic relations between ontologies of the same type, such as between two social ontologies of Alice and Bob or between two object ontologies or even permission ontologies, should provide a way to reuse (propagate) the permissions assigned by existing rules. For instance to reuse the rules assigning permissions to 'VIP' users of Alice onto Bob's 'Senior' customers.

7.2 Semantic Matching

Semantic Matching (S-Match)[38, 39, 40, 41] is a tool to find the semantic relations between nodes of two lightweight ontologies. The original idea is

Table 7.1: Semantic Matching on Labels

	$Friend^3$	$Business^7$	$Product^1$	$Apple^3$	$Lenovo^1$	$Soccer^1 \sqcap Fan^2$
$Sale^2$	$\perp$	$\sqsubseteq$				$\perp$
$Digital^3$						
$Laptop^1$			$\sqsubseteq$			
$MacBook^1$				$\sqcap$	$\perp$	
$Thinkpad^1$				$\perp$	$\sqcap$	

to calculate the semantic similarity such as *equal, overlapping*, etc. between the categories of the two given classifications. The core of a S-Match procedure consists two rounds of matching on the *concepts at label* and *concept at node* as described in [41]. Semantic similarities are defined with sense relations. *Equal* '$\equiv$': one concept is equal to another if there is at least one sense of the first concept, which is a synonym of the second. *Overlapping* '$\sqcap$': one concept is overlapped with the other if there are some senses in common. *Mismatch* '$\perp$': two concepts are mismatched if they have no sense in common. *More general / specific* '$\sqsupseteq, \sqsubseteq$': one concept is more general than the other if and only if there exists at least one sense of the first concept that has a sense of the other as a hyponym or as a meronym.

Semantic matching allows to clarify the semantic relations between the two lightweight ontologies. Given enough background knowledge (e.g., $MacBook^1$ is a $Laptop^1$ and a $product^1$ of $Apple^3$), we can find the semantic similarities between the two ontologies in Figure 6.2 and Figure 6.3 as shown in Table 7.1. Here we can see, for instance, that $Sale^2 \sqsubseteq Business^7$, $Friend^3 \perp Sale^2$ and so on, where an empty cell means that the system is not able to find a semantic relation.

WordNet [63] is an often used background knowledge base in which possible relations between senses (meanings of word) are provided. But for special domains such as eBuseness or web blogs, corresponding background knowledge should be used as the knowledge base.

7.3 Automated Rule Generation in *RelBAC*

As we have pointed out that it is challenging and error-prone to manage an access control system described in the previous sections. How to use this kind of semantic similarities to improve the situation?

RelBAC provides reasoning ability about the knowledge base such as consistency checking and query answering such that *membership checking, security property enforcement* can be reasoning tasks at design time and *access control decision* can be reasoned at run time as described in [88]. An access control system for a larger and more complex eBusiness solution still needs some help to generate access control rules e.g., when the vendor is not expert in access control, as it is the most often case with social networks, or in the case of relatively large and complex ontologies or highly dynamic policies. We can find with S-Match [40, 39, 38] that there exists similarity between subject and object lightweight ontologies although they may be heterogeneous ontologies built independently. This will help to generate candidate permissions to be submitted to the user for approval, or generate semantically motivated constraints between subject and object categories, and so on.

7.3.1 Rule Design

For any access control systems, the stage of rules design is very important because a cute rule set will simplify later work of enforcement and management. Semantic Matching between subject and object ontologies will clarify all the semantic relations between categories of the two lightweight ontologies. For example, given the background knowledge about the relations $MacBook^1$ is a $Laptop^1$ as a $product^1$ of $Apple^3$ etc., we can find the semantic similarities as list in Table 7.1. As WordNet does not 'know' the word as 'MacBook', which is common with the birth of many new words in this Information Age, we should enrich the knowledge base with the facts such as '*Apple*^{3} *is a IT company selling digital products such as MacBook and IPod.*'.

From Table 7.1, we can see the semantic similarities such as $Sale^2 \sqsubseteq Business^7$, etc. What is interesting in this example is that the relation between $Friend^3$ and $Sale^2$ is $\perp$ i.e. the two words has nothing in common. It is weird but true because $Friend^3$ means '*a person with whom you are acquainted*' and $Sale^2$ is '*the general activity of selling*'. This will be the general case when we try to match a subject ontology with an object ontology. If we went on with the second step of a classic S-match algorithm, i.e. to match the concepts at node in the lightweight ontologies, $\perp$s will fill up the table. Thus, a table full of '$\perp$' provides no suggestions for rule design or reuse. Therefore, we only use the first round S-Match to discover the relations. These relations provide the following suggestions to in rule design.

Semantically Related The cells marked with '$\sqsubseteq, \sqsupseteq, \equiv, \sqcap$' represent the semantic similarity of the corresponding concepts. It is reasonable to assign corresponding users the right to access the objects. For example, the relation $Sale^2 \sqsubseteq Business^7$ suggests that some access right, say $Read$, should be assigned to the $Business^7$ $Friend^3$ to some $Sale^2$ categories. It is obvious here in the small example ontologies. However, facing a large eBusiness such as Amazon.com, these similarities will be very useful for the administrators.

Explicit Unrelated The cells marked with '$\perp$[1]' represent that the corresponding concepts are found 'unrelated' in the knowledge base. We

[1]Here we shorten the axiom '$C_1 \sqcap C_2 \sqsubseteq \perp$' as '$C_1 \perp C_2$'.

have to differentiate the real world semantics of these '$\perp$'s.

- $Sale^2 \perp Friend^3$ is a *subject/object mismatch* only because they are referring to activity and person respectively and background knowledge do not contain similarity between different types;
- $MacBook^1 \perp Lenovo^1$ comes from that '*MacBook is a product of Apple company*' and '*Apple and Lenovo are different IT companies*'. This explicit mismatch relation between the two concepts suggests that no access should be assigned, even an explicit prohibition should be assigned;
- $Sale^2 \perp (Soccer^1 \sqcap Fan^2)$ covers both of the upper cases so it is still explicitly unrelated and no access should be assigned.

I don't know (IDN) The blank cells of the table mean that the knowledge base does not know any existing relation between the corresponding concepts. Then it is up to the administrator to decide whether to assign some access or not. From Table 7.1 we can see that this kind of cells are the majority because the knowledge base we use is not specially for eBusiness domain and if it is specially enriched with domain knowledge, more semantic relations will be found and more suggestions will be offered.

7.3.2 Rule Integration

One important evolution of subject and object ontologies is to integrate similar ontologies. For example, an eBusiness vendor will enlarge her social network to involve more customers and very likely to integrate the customer ontology of another vendor, or similarly to integrate another goods ontology. The classical access control solutions require the administrator to re-design new rules for these evolving parts. Even for the similar ontologies, all assignments have to be made once more. For example, the

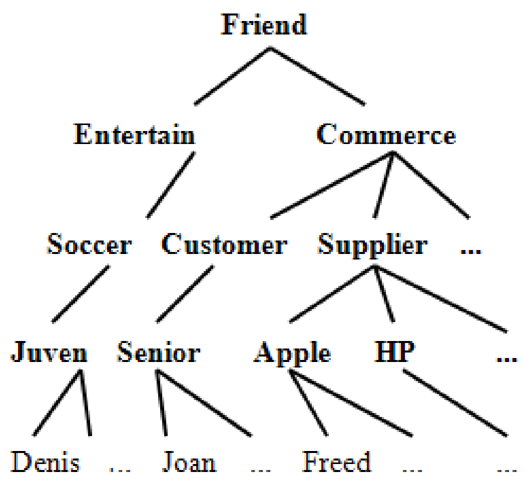


Figure 7.1: Bob's Ontology

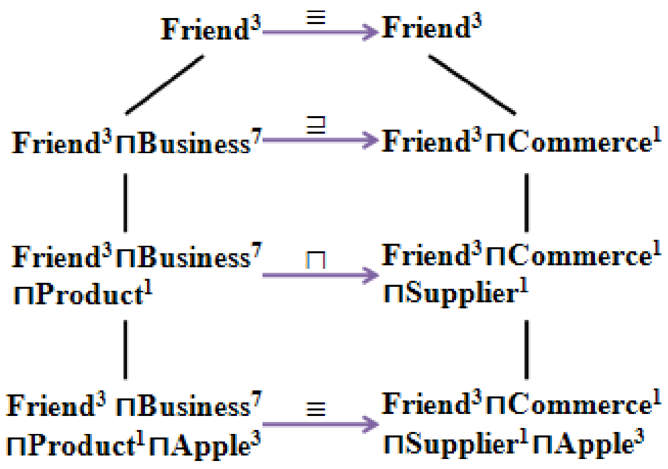

Friend³
≡
Friend³
Friend³⊓Business⁷
⊒
Friend³⊓Commerce¹
Friend³⊓Business⁷ ⊓Product¹
⊓
Friend³⊓Commerce¹ ⊓Supplier¹
Friend³ ⊓Business⁷ ⊓Product¹⊓Apple³
≡
Friend³⊓Commerce¹ ⊓Supplier¹⊓Apple³

Figure 7.2: Ontology Matching for Rule Reuse

vendor in the scenario of Chapter 6 would like to merge another ontology of subjects as Bob's Social Ontology as Figure 7.1. A customer set called 'Senior' has the similar intuition to the 'VIP' set in the previous ontology.

The resulting semantic relations can be used along the lines of what described in the previous sections either to drive the merging of the two ontologies or to create mappings. The mappings in turn can be used for the propagation of permissions from one ontology to the other. Thus for instance the system administrator might enforce that the equivalence mapping between the two root nodes in Figure 7.2 means that a *Read* permission on the left root node propagates to the right root node. These kinds of mappings are very similar to the C-OWL mappings introduced in [12] and should be used whenever a full merge of the two ontologies is not advisable or there are good reasons to keep the two ontologies distinct.

We show in Figure 7.2 the results of S-Match on two branches of the 'friend' ontologies in Figure 6.2 and 7.1. The semantic similarity axioms can be added to the knowledge base of access control and the rule reuse is done without further efforts. For example,

$$\{(Friend^3 \sqcap Commerce^1) \sqsubseteq (Friend^3 \sqcap Business^7),$$
$$Business^7 \sqsubseteq \alpha\} \models Friend^3 \sqcap Commerce^1 \sqsubseteq \alpha$$

So any *subject-centric* rules that assigned to $Business^7$ permissions will also propagate to $Friend^3 \sqcap Commerce^1$ without creating new rules for the new subject sets. Similar reuse applies on objects as well when S-Match is used to find the semantic similarities of the object ontologies.

7.4 Summary

Based on the RelBAC formalization of the access control problem in social networks, we can organize users, objects and permissions as (lightweight) ontologies. This allows to represent access control rules and policies as DL formulas and to reason about them using state of the art off-the-shelf reasoners. However, when the knowledge base becomes more and more complex, the rule management tasks explode. Thus it requires automated tools to help design and reuse rules.

In this chapter, we have shown how it is possible to use semantic matching technology to discover and exploit the underlying semantic relations between subject and object ontologies and between two user (or object) ontologies. The resulting automated reasoning capabilities can be exploited to support the system administrator in policy management, which is time consuming and error-prone before.

Chapter 8

Evaluation

RelBAC is not only a model in theory, but applicable in real access control systems. We will show in this chapter some evaluation results to prove the applicability of the model.

At the beginning of the chapter, we analyze the state of the art DL reasoners and choose Pellet [77] for *RelBAC* based on tests in Section 8.1; then, we describe the algorithm to build benchmark ontologies in Section 8.2; later, in Section 8.3 we test the reasoner performance on different types of permission assignments and different number of individuals; at last we make a conclusion in Section 8.4.

8.1 Reasoner Selection

The *RelBAC* model lies in a set of domain specific Description Logics (on different access control problems). Design time and run time administration are automated by reasoning services such as hierarchy and membership managements, permission assignment administration, security property enforcement, and access control decision. Although theoretically any off-the-shelf DL reasoner can be used for *RelBAC* reasoning, there are still some general requests to be satisfied.

- From a theoretical point of view, *RelBAC* describes all the access control items, users, hierarchies, rules as description logic formula and can be stored as a standalone OWL file. The size of the file may be as large to tens of megabytes. The ability to process large files should be optimized according to the possible number of access control policies.

- Close-world assumption [73] should be supported or easily added, at least on access control discipline. As we know that classic DL follows an open world assumption in contrast to a database schema which assumes a closed world. But from a security point of view, those not explicitly permitted access should be denied. It is practical that the administration cannot predefine all the cases of deny. Thus, closed-world assumption, at least for the access control decision reasoning, should be enforced for *RelBAC*.

- The reasoner should be well structured and clearly documented so that important parts can be cut and fit to a real *RelBAC* system. The trade-off between expressiveness and efficiency is always important for a real system and that is why many access control systems turn for Prolog based logical frameworks such as [9, 52, 20, 57, 6] and so on. Thus, since we chose DL as the basic logical framework, we have to consider in an access control point of view to achieve real time reasoning with reconfiguring and optimizing the reasoner.

There are many choices of DL reasoners. Among the most famous are developed by University of Manchester (FaCT++[79], KAON2[64], MSPASS[51]), Mindswap Lab of University of Maryland (Pellet[1][77]), University of Dresden (CEL[3]) and people originated from University of Hamburg (RacerPro[49, 50]).

[1]Now supported by Clark & Parsia, LLC.

Table 8.1: DL Reasoners

	Language	**Interface**	**Algorithm**	**Code**	**Support**	**License**
FaCT++	SHOIQ with small datatypes	OWL DIG	Tableau	C++	Univ. of Manchester	Open
KAON2	SHIN without large cardinality	OWL RDF	Reduce SHIQ to datalog program	Java	Univ. of Manchester	Open
MSPASS	ALBO	DFG	SPASS	C	Univ. of Manchester	Open
Pellet	SHOIQ with small datatypes	OWL DIG Jena	Tableau	Java	Univ. of Maryland and Clark&Parsia,LLC	Open
RacerPro	SHOIQ	OWL DIG Lisp	Tableau	Lisp	Univ. of Hamburg	Semi-

We have compared these reasoners in Table 8.1. From the table, we find that Fact++ and Pellet supports SHOIQ (partly) and convenient interfaces, and they are both open-source, tableau-based reasoners supported by famous groups. So they are enrolled to the second round.

8.2 Benchmark Creation

As has been discussed in Chapter 4, *RelBAC* may have a bunch of reasoning tasks such as *knowledge base consistency checking, hierarchy management, membership management, permission assignment, security property enforcement, etc.* A *RelBAC* knowledge base consists of two parts: $\mathcal{P}$ access control policies with respect to the DL TBox; $\mathcal{S}$ access control states with respect to the DL ABox.

As we plan to use the reasoner for design time and run time reasoning as discussed in Chapter 4, the benchmark ontologies should have the following features.

- Thing as the root of the ontology with 100+ individuals;
- Everybody as the root of the user subtree with 10 layers and about 100 subject groups (DL concepts);
- Everything as the root of the object subtree with 9 layers and about 100 object classes (DL concepts);
- 3 properties (for permission as DL roles) forming a 2-layer property hierarchy;

We have not found existing benchmarks for access control. Therefore, we create a script to build this kind of benchmarks on the fly based on Algorithm 5. In the algorithm, Line 3-5 are used to build subject, object and permission hierarchies (trees) according to input parameters. Then, for each subject and object node, the algorithm creates owl concepts on Line 7. Line 9-12 randomly generate individuals for the concepts. Last but not least, Line 13-18 randomly create a number of permission assignments according to the input 'NAss' (number of assignments).

The intuition here is just to add new lines with owl tags to a standard OWL [69] file. All the relations between nodes are generated randomly into tree-like hierarchies. The OWL restrictions are used to connect user group and object class (referred as 'permission' in the algorithm, with 4 forms such as 'some', 'all', 'min', 'max'). There is a shortcoming of this algorithm: the random chosen restriction in ontology might lead to inconsistency. The more restrictions added to the ontology, the ontology appears more probably inconsistent.

Algorithm 5: BuildOnto1.0

```
Data: Tree: struct tree
Input: Int NUser, Int NObj, Int NPer, Int NInd, Int NAss
Output: an owl file
File ac = new File AC-OWL.owl;
Add to ac the common headings of owl;
Tree TUser = new Tree(NUser);
Tree TObj = new Tree(NObj);
Tree TPer = new Tree(NPer);
for each node in TUser, TObj, TPer do
    Add to ac a new corresponding concept;
end
for Int i = 0; i<NInd; i++ do
    Random choose a node n from TUser or TObj;
    Add to ac a corresponding individual $ind_i$ under n;
end
for i = 0; i<NAss; i++ do
    Random choose a node np from TPer;
    Random choose a node nu from TUser;
    Random choose a node no from TObj;
    Add to ac the permission assignment of np from nu to no;
end
return ac;
```

8.3 Results

In this test, we choose consistency checking as the representative of all the reasoning service provided by the reasoners. The testing results are shown in 3 categories. First, we get a set of general purpose ontology: the ‘Wine’ ontologies from W3C.org and test the performance of the two DL reasoners enrolled to the second round, which are FaCT++ and Pellet. Then in order to test the reasoner performance on each type of assignment, we create benchmark ontologies for each type of rules (‘some’, ‘only’, ‘min’ and

Table 8.2: Wine Ontologies

Wine Series	**Wine_ 0**	**Wine_1**	**Wine_2**	**Wine_3**	**Wine_4**	**Wine_5**	**Wine_6**
Size(Byte)	88928	144526	279963	254098	308884	363670	642230
Class No.	142	141	141	141	141	141	141
Property No.	13	13	13	13	13	13	13
Ind. No.	162	483	805	1127	1449	1771	3381

'max') with random number of triples. Finally, we test the correspondence between individual numbers and the time consumed by the reasoner.

These tests have been done on a PC with Intel Pentium D 3.00GHz processor, 4.00GB RAM (2G*2), 160GB HDD. We use Protégé 4 as the platform to load in the ontologies.

8.3.1 Performance Test

We test the performance of FaCT++ and Pellet with the wine ontologies as list in Table 8.2. The results of the tests are in Table 8.3. 'F' and 'P' stand for FaCT++ and Pellet respectively. In the reasoner synchronization stage, the reasoner cleans the knowledge base, converts the ontology to DIG api and updates reasoner; during consistency checking, the reasoner prepares a query in <0.001 ms, checks the consistency and updates the ontology in the platform. We can see from the evaluation results the following:

- With the ontology scale increasing, the time for checking the consistency plays a larger and larger part in the total time. FaCT++ takes more time than Pellet on this section.

- The time consumed for consistency checking increases monotonously but not proportional to the size of the ontology. For the first 4 tests, the time spent by FaCT++ increases faster than by Pellet which means Pellet spends less time to check the consistency of the same complexity.

- Through the first 4 tests, we find that Pellet performs better in contrast to FaCT++ for larger number of individuals in the benchmark ontologies. But both reasoners do not perform well with large number of individuals. For the test on Wine_6, Pellet throws a 'JaveHeapStack' error while FaCT++ successfully finishes checking although quite a lot of time has been used.

- The complexity of wine ontology series increases only on the individual number, but not in concept or property. This does not require more expressive DL language which means both two reasoners can reason about ontologies of this DL language (ALHQ).

As shown in Table 8.2, the complexity increase is only due to the number of individuals involved in the ontology. So the test means that given a 'computable' number of individuals, Pellet performs better than FaCT++. So we choose Pellet as the reasoner for further *RelBAC* evaluation.

8.3.2 Single Type Assignment

Here, we list the results of consistency checking on one single type of assignment: 'some', 'only', 'min' and 'max'. The cardinality of 'min' and 'max' permissions are randomly chosen among integers from 1 to 50.

'some' we have the test on 10 randomly built ontolgies with 1 to 50 'some' permissions assigned. The time to check the consistency of the ontology does not follow any regular proportion. The average time spent on consistency checking for 5 ontologies with 545 to 775 triples varies from 9 to 136 milliseconds.

Table 8.3: FaCT++ vs. Pellet

Wine		**Synchronize Reasoner**			**Check Consistency**			**Total**
Reasoner		**Clean**	**DIG**	**Update**	**Prepare**	**Check**	**Update**	
Wine_0	F	0.016	0.125	0.078	<0.001	6.735	0.016	7.031
	P	0.016	0.124	0.082	<0.001	1.093	0.016	1.359
Wine_1	F	<0.001	0.266	0.156	<0.001	62.031	<0.001	62.547
	P	<0.001	0.265	0.172	<0.001	8.828	0.032	9.375
Wine_2	F	0.015	0.453	0.25	<0.001	181.375	<0.001	182.125
	P	<0.001	0.442	0.328	<0.001	23.36	0.031	24.188
Wine_3	F	0.032	0.672	0.344	<0.001	369.75	0.032	370.907
	P	<0.001	1.031	0.406	<0.001	46.438	0.015	47.922
Wine_4	F	0.031	0.797	0.688	<0.001	624.656	0.031	626.281
	P	0.015	0.969	0.891	<0.001	77.625	0.047	80.015
Wine_5	F	0.032	0.984	0.453	<0.001	914.203	0.031	915.782
	P	0.001	0.922	1.484	<0.001	2108.438	0.031	2111.406
Wine_6	F	0.016	1.781	1.172	<0.001	3323.953	0.016	3327.313
	P	0.016	1.75	1.125	<0.001	-	-	-

'only' we have the test on 10 randomly built ontolgies with 1 to 50 'all' permissions assigned. The time to check the consistency of the ontology does not follow any regular proportion. The average time spent on consistency checking for 5 ontologies with 570 to 730 triples varies from 15.8 to 119 milliseconds.

'min' we have the test on 10 randomly built ontolgies with 1 to 50 'min' permissions assigned. The time to check the consistency of the ontology does not follow any regular proportion. The average time spent on consistency checking for 5 ontologies with 570 to 828 triples varies from 47.6 to 261.6 milliseconds.

'max' we have the test on 10 randomly built ontolgies with 1 to 50 'max' permissions assigned. The time to check the consistency of the ontology does not follow any regular proportion. The average time spent

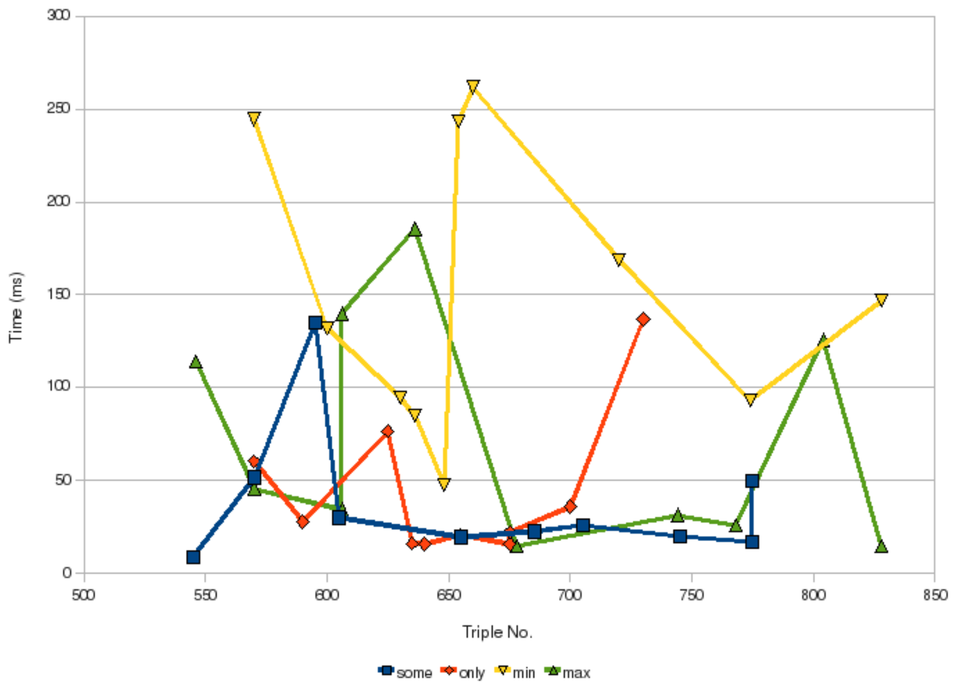


Figure 8.1: Reasoner Performance on Single Type Assignment

on consistency checking for 5 ontologies with 546 to 828 triples varies from 14.6 to 185.4 milliseconds.

As shown in Figure 8.1, these automatically generated benchmark ontologies are simple and easy to check (less than 300 ms). The reasoning tasks can be achieved in 'real' time.

8.3.3 Multiple Type Assignment

Here we list the results of consistency checking for ontologies with multiple type assignments: 'some'+'only', 'min'+'max' and 4 types as is shown in Figure 8.2.

'some'+'all' we have the test on 50 randomly built ontolgies with 1 to 50 'some' or 'all' permissions assigned. The time to check the consistency of the ontology doesn't follow any regular proportion. The average time spent on consistency checking for 5 ontologies with 540 to 785

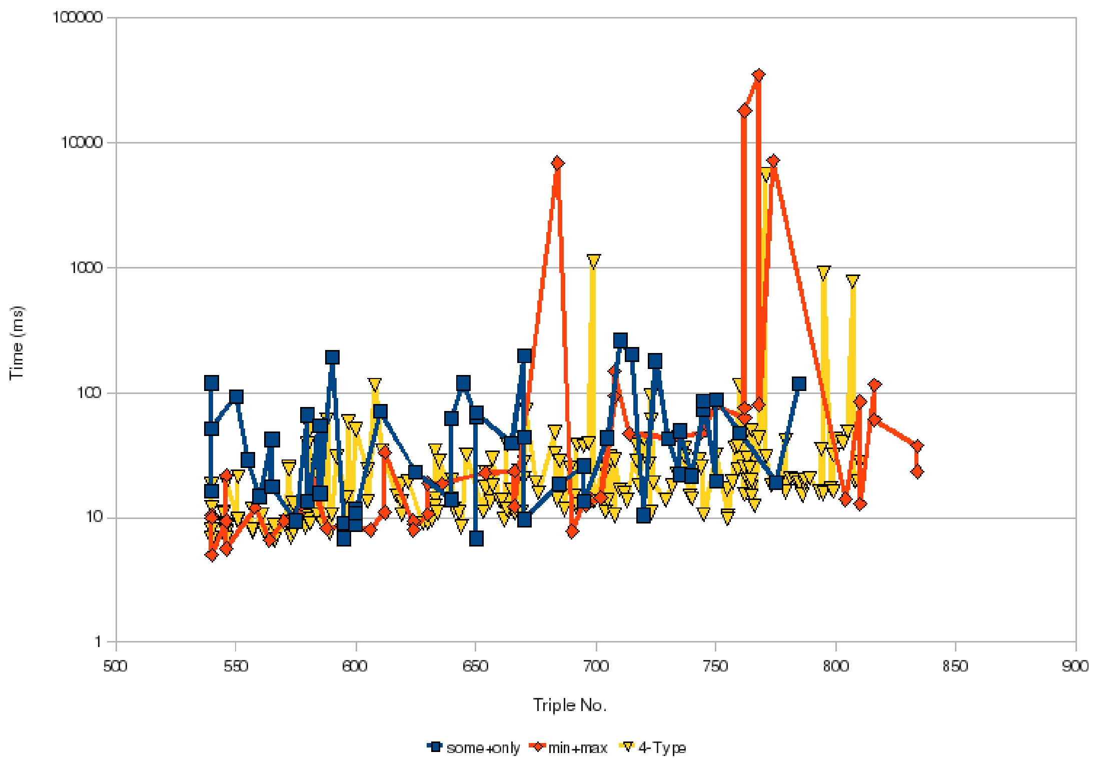


Figure 8.2: Reasoner Performance on Multiple Types Assignments

triples varies from 14.6 to 185.4 milliseconds. The best and worst cases are 5 and 646 ms.

'min'+'max' we have the test on 50 randomly built ontolgies with 1 to 50 'min' or 'max' permissions assigned. The time to check the consistency of the ontology doesn't follow any regular proportion. The average time spent on consistency checking for 50 ontologies with 540 to 834 triples varies from 5 to 35192.6 milliseconds. The best and worst cases are 4 and 35804 ms.

4 Types we have the test on 188 randomly built ontolgies with 1 to 50 'some', 'all', 'min' or 'max' permissions assigned. The time to check the consistency of the ontology doesn't follow any regular proportion. The average time spent on consistency checking for 188 ontologies with 540 to 810 triples varies from 6.6 to 5558.2 milliseconds. The best and worst cases are 5 and 6414 ms.

The tests results highlight that:

1. Although some of the tests (2%) appear abnormally time-consuming (35804ms to the most), the rest cost acceptable time even we expect a ‘real-time’ response.

2. The worst and average cases time consumed for ontologies with just ‘min’+‘max’ is longer than the 4 type included ontologies. From the previous tests, ‘some’+‘only’ assignments cost far less. we may conclude that the cardinality related assignments are relatively time consuming.

3. As the assignments in each ontology are randomly generated, the complexities of each ontology should be similar and the time consumed for consistency checking should be relatively similar. In the real case, the triple number does not affect the checking much. There are 2% ontologies that cost exceptional long time to check the consistency. After exploring into the details of the ontologies, we find that the ‘min’ and ‘max’ rules with large cardinality, say above 40, are the main reason for this kind of long time consuming.

8.3.4 Increasing Individual

To test the impacts of individual number on the consistency checking, we create demo ontologies with 20 subject concepts, 20 object concepts, 10 permission properties, 20 assignments and increasing number of individuals (125, 250, 500, 750, 1000). Algorithm 5 is reused to create these ontologies with random trees for subject, object concepts and permission roles.

In Figure 8.3, we see 5 kinds of points connected with lines. Each point corresponds to the average time of 5 tests spent on consistency checking

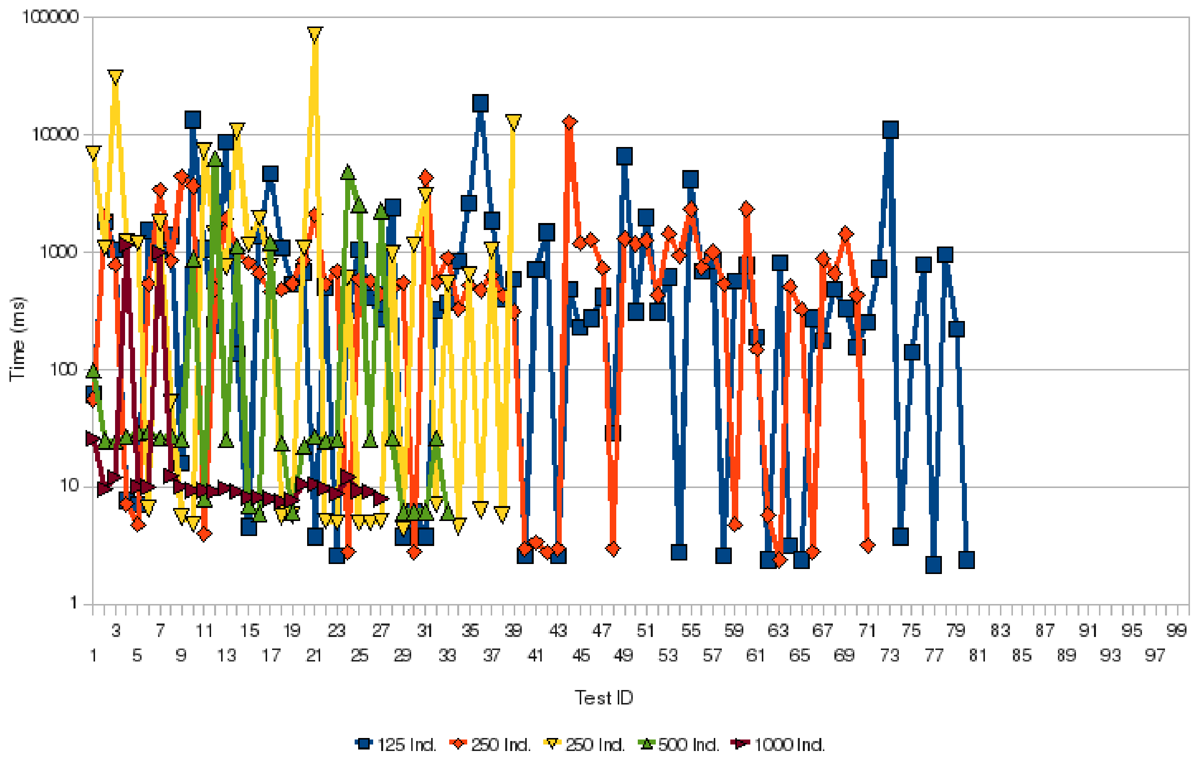


Figure 8.3: Reasoner Performance on Increasing Individuals

of an ontology created as above. Each type of points are connected with colored lines to show the variation among exactly the same ontology with the same number of arbitrary permission assignments. The figure shows that,

- The time consumed for tests on different permission assignment (given other conditions remain the same) is not stable. This means that the parameters of the assignments, such as the numbers chosen for 'min' and 'max' rules play an important part in the checking. The ontology with rules of larger numbers tend to be more time consuming.
- The time spent for each kind of ontologies (with increasing individual numbers while other conditions the same) does not increase with ascending individual number. This means that the individual number is less important than the number of 'min'/'max' assignments.

- The line of larger individual numbered ontologies is 'shorter'. As we test 100 ontologies of each kind, the lines should be connections of 100 points. But some of the tests throw 'OutOfMemory' error that terminate irregularly and in such test, no checking results (true or false) or time consumed is given so in these cases, no points are drawn on the figure. The portion of such errors increase with the accumulating individual numbers. When we run with larger memories for the Java VM, the tendency remains.

- A symptom shown in the testing results (not shown in Figure 8.3) is that the larger number of individuals in the ontologies, the ontology has more possibility to be inconsistent. This is coherent to the intuition that the 'min' and 'max' rules are more easily violated with larger number of individuals. This means with the growth of the knowledge base, it tends to be more error-prone which is a good reason that automated tools should be provide for the new rule design or existing rule reuse.

8.4 Summary

As a conclusion, we find the following out of the testing results.

1. Out of many off-the-shelf reasoners, we choose Pellet [77] as the one for *RelBAC*, based on its performance in contrast to other reasoners such as FaCT++ [79].

2. The reasoner performance is acceptable and especially good in most the cases for the 'some' and 'only' rules. The increasing individual number costs more memory, and the time for consistency reasoning.

3. In the bad cases, the arbitrarily assignments with number restrictions are challenging for the reasoners and we should adjust the reasoner

on this or not use less ‘min’/‘max’ rules in real cases.

4. There are many practical issues left for implementation stage such as the Closed World Assumption [73].

In conclusion, the test results are encouraging and challenging. We see the ability of *RelBAC* on automated reasoning from a quantity point of view in addition to the quality view in Chapter 4. However we still have much to do as these general purpose DL reasoners, no matter Pellet, FaCT++ or any others, are not specially designed for the domain of access control, not to say for *RelBAC*. Thus adjustments should be taken to meet the practical needs in *RelBAC*.

Chapter 9

Implementation

The implementation of a real *RelBAC* based system is non-trivial, because the access control component of a system has the risk to be the bottleneck as every request will have to be authorized by the access control before execution.

In this chapter, we present the system architecture of a typical *RelBAC* based access control system (Section 9.1). Moreover, we analyze the necessity and practical solution for the closed world assumption (Section 9.2) and the implementation of the 'All' rule (Section 9.3).

9.1 System Architecture

A system based on *RelBAC* includes three parts, the user interface to deal with input and output; the knowledge base consisting of access control policy specifications ($\mathcal{P}$) and access control state specifications ($\mathcal{S}$); and the reasoner, used as a blackbox which can reason about knowledge base and queries.

Here we present the architecture of a system implementing the *RelBAC* model in Figure 9.1. In general, hollow arrows stand for user related operation or information exchange; solid arrows represent internal data flow and interaction. From the administration point of view, Arrows (a) and

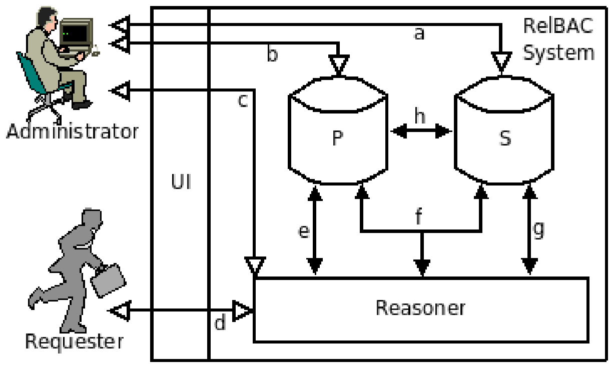


Figure 9.1: The System Architecture using RelBAC.

(b) are direct queries and updates to the knowledge base, where Arrow (h) represents the interaction of knowledge between $\mathcal{P}$ and $\mathcal{S}$. The design time reasoning services, which allow to perform redundancy checking, conflict checking, etc. are offered to the administrator by Arrow (c). Arrow (e), (g) and (f) stand respectively for TBox reasoning, ABox reasoning and TBox+ABox reasoning. From a requester point of view, a query for permission is interpreted by the UI and handed to the reasoner through Arrow (d). The reasoner processes the query as an entailment reasoning task with respect to the knowledge base through Arrow (e), (f) and (g) to provide access control decisions at run time.

Our implementation integrates an open source reasoner Pellet [77] through owl-api. Pellet is developed in Java and easy to be integrated into large software systems. The state of the art Pellet 2.0rc4 offers necessary reasoning ability for *entailment, consistency, subsumption,* and *realization*. The incremental reasoning about ABox updates in Pellet is also a nice feature for *RelBAC* as the run time membership update relies on ABox changes and reasoning.

We have built some ontologies using RDF/OWL which describes the scenario of the SFA [82] example in Section 4.1.2. The sample policies listed in Chapter 4 were also included. To avoid biases, we randomly

Table 9.1: Policy Base Consistency Test

	Size	Set	Rule	Ind.	Time(ms)
1	61.3	14	10	426	48.0
2	86.8	141	131	162	592.0
3	141.1	141	131	483	2191.0
4	273.4	141	131	805	5677.0

generate groups and classes of hundreds of individuals. The results are listed in Table 9.1.

A small business with 5 user groups, 9 object classes, about 400 individuals (including employees and documents) with 10 access control rules can be formalized in an ontology of 61.3KB. It takes less than 50 ms to complete consistency checking as is shown by the first record of the table. When the business grows ten times in sets (either group or class) numbers and in rules numbers (as record 2), the checking time grows too. Even if we just increase the number of individuals randomly for users or objects, the time consumed grows exponentially. Although preliminary, the tests show that the chosen reasoner does not scale well to real access control problems. There is still much work to do for a real-time access control system.

9.2 Closed World Assumption

Closed World Assumption (CWA) is about the completeness of the knowledge base. R.Reiter discussed this problem based on the database schema in [73]. It is intuitively to assume the 'closed world' for many common sense schemas such as in a database scenario, the records form a 'complete' knowledge that other information not in the records is considered none existing. On the contrary, Description Logic [4] assumes an 'open' world, which means that the knowledge bases are assumed incomplete such that the short of information will not necessarily lead to a negative reasoning result.

RelBAC is formalized in a DL based logical framework so the semantics of a *RelBAC* knowledge base do not support the CWA. This is fine with a common DL based system, but not OK for *RelBAC*. An obvious and intuitive reason is that from a security point of view, the access control decision (mentioned in Section 4.2) cannot follow the open world assumption and offer an answer as 'I don't know'. As is pointed out in the basic access control decision algorithm in Section 1.1, if no knowledge supports 'deny', neither for 'permit', the system should reject the access request.

To the best of our knowledge, there is no DL reasoner that supports full CWA though some of them are declared to do so partly.

We see at least 2 kinds of closed world requests for *RelBAC*.

Access Control Decision For security reasons, a non-explicitly permitted request should be denied. This means only when the request query is the reasoning result, otherwise it is denied by the system.

To achieve this, when we use the reasoner for access control decision via the information flow as denoted by the arrow (d) in Figure 9.1, the query answer of the reasoner should not be returned directly to the requester. An 'interpretation' layer should be added to achieve Algorithm 1. When the reasoner does not give explicit answer of 'permit', the layer should get output information of the reasoner and interpret it to the requester, for example to tell the user that the reason of deny is that the requester is in a group that restricted to perform such access.

Membership Management An integrity constraint of user membership is a good motivation of CWA. For instance,

$$Friend \sqsubseteq \exists Read.Blog \sqcup \exists Upload.Video$$

is not sufficient to declare that a friend is allowed to either read some blog entries or upload some videos. Actually any knowledge base with $Friend(ann)$ is a model of this concept. But it does not necessarily tell whether Ann can read blog or upload video. Here, $\exists Read.Blog$ and $\exists Upload.Video$ are two virtual user groups that Ann should be a member of either (or both) of them, but cannot be a member of neither. Thus, in the membership management, *RelBAC* should explicitly claim in the $\mathcal{S}$ about all the memberships of individuals. For the same example, additional axioms such as

$$\{ann, bob\} \sqsubseteq \exists Read.Blog \text{ or } \{bob, cate\} \sqsubseteq \exists Upload.Video$$

should be designed.

This is applicable for not only subject, but object and permission as well. To 'achieve' CWA, in a DL based logical system, such as *RelBAC*, such kind of knowledge should be explicitly declared.

We can support CWA to some extend in *RelBAC* by applying the proposals above. It is good enough for now, and we are trying to find new solutions when new problem arises.

9.3 'All' Rule Implementation

The 'All' rule has been discussed in Section 3.5 as rules (3.24–3.27). It is an important kind of rule because it facilitates administration by simplifying the numerous rules on Cartesian products of the subject and object sets into one single rule. Moreover, this is the classical way to assign access from a subject set to an object set before *RelBAC* appears. People are familiar with such assignments. Although *RelBAC* provides far more fine-grained access control rules, among which the cardinality related rules have never been supported before, this kind of classic assignment should be supported.

As shown in Theorem 1, the ‘All’ rule refers to the combination use of complement constructor of both role and class. Generally speaking, class complement is fairly easy to achieve with a universal concept. But the complement of role is rather difficult. It is regarded by [26] as a ‘complex’ role which may cause undecidability of the corresponding DL language. Intuitively, the ‘All’ rule defines a set of elements that each has a relation with another given set of elements. In contrast, the original ‘universal value restriction’ in DL defines that set of elements that each has a given relation only with another set of elements. Well, the latter can be traced through the given relation and secured that the target element is in a certain set but the former faces the problem to search for all the relations and then to find the complements of the given relation. This is almost impossible as a reasoner cannot easily get a ‘universal’ role. So it is theoretically possible to have ‘All’ rules represented in classic DL extended with class and role negations, but it is hard to reason about ‘All’ rules in practice.

How to solve this problem in the real life? Actually, we encode the *RelBAC* knowledge base with OWL-DL language and use the OWL-API for Pellet. SWRL [78] language is an acceptable OWL-API for Pellet too. We see that an ‘All’ rule can be encoded into a SWRL rule. Thus, the reasoner can infer with the SWRL rules instead of using role negation.

For example, the following ‘All’ rule

$$Friend \sqsubseteq \forall Blog.Read$$

will be encoded as a SWRL rule as follows.

$$Friend(?x) \wedge Blog(?y) \rightarrow Read(?x, ?y)$$

Thus through the OWL-API, these rules can be encoded into the *RelBAC* ontology and dealt by the DL reasoner.

9.4 Summary

In this chapter, a typical *RelBAC* system architecture has been proposed to illustrate the inter-operation between parts of the components. Several practical issues on *RelBAC* implementation are also studied here such as the Close World Assumption and the realization of the 'All' rules. In our preliminary implementation of *RelBAC*, we built a basic system with only singleton groups and 'All' rules were replaced with 'some' rules. In the future, we will test the way to replace 'All' rules with SWRL format. As for closed world assumption, we use the 'owl:one-of' constructor to enumerate all the known individuals for the membership management and use an interpretor layer to achieve the default 'deny' decision. There are still many practical problems for implementation that we'll discover and solve in the future.

Chapter 10

Conclusion

The new Web 2.0 solutions bring the web to people regardless of age, gender or interest. The upcoming era of the glorious web allows end users not only to read and download but also to publish and share. Moreover, people seem to neglect the problem of privacy, that we lose control of these data we put online. Scholars of computer science come to realize this specific problem of access control, and accordingly many efforts are made to apply existing access control models in these recently emerging scenarios. However, the fast developing web technologies bring new challenges wherein existing static models, such as RBAC, cannot formalize the dynamic subjects and objects, not to say the context changing related permissions.

In the thesis, we have developed a novel access control model named *RelBAC* to face these new challenges. In addition, *RelBAC* is formalized with access control domain specific Description Logics. In *RelBAC*, a permission is a binary relation of an Entity Relationship model, and formalized as a Description Logic role. This feature makes it possible to control the access in fine-grained cardinality.

The advantage of using a logical framework for an access control model is the automated reasoning ability of the logic. The logical formalization of *RelBAC* simplifies the access control management into automated reasoning provided by any off-the-shelf reasoner such that at design time, the reasoner can help to check redundancy and conflicts in hierarchy and membership management, enforce security properties of separation of duties; at run time, the reasoner can answer the access control request and help to make access control decisions.

RelBAC is not a model generated on the fly, as it has been inspired by many existing models such as AM, RBAC etc. Thus it inherits their advantages such as object-oriented rules from AM and subject hierarchies from RBAC. It can be adjusted to support these models by small extensions. We have shown in the thesis, how to adjust *RelBAC* to support modern access control models such as RBAC.

As an access control model for the web applications, *RelBAC* model can take use of many semantic web technologies such as lightweight ontology and semantic matching. The knowledge base can be encoded into lightweight ontologies so that the permission can propagate through the unified relations in the hierarchies. Semantic matching can help to find the similarities between subject and object and suggest for new access control rules. It can also help to reuse existing rules with the similarities found between subjects or between objects. We have also shown in this thesis these possibilities and details to apply these tools on *RelBAC*.

In this thesis, some evaluations have been performed in order to choose a proper reasoner and to understand the weights of parameters such as the number of individuals or the different types of permission assignments. We have implemented the basic idea of *RelBAC* and analyzed several practical issues for the implementation.

Among several open issues, we intend to look at the following in the future.

- The complexity of security problem formalization is still not clear and the theoretical work to represent and analyze corresponding problems in *RelBAC* will solid the foundation of the model and improve its popularity.

- We have proposed partial solutions to some issues such as Closed World Assumption in the access control domain. However, there is still systematical analysis work in this direction.
- The full implementation of *RelBAC* and the adjustment of general purpose reasoners to fit the access control domain still need efforts.

In brief, *RelBAC* is a novel, expressive and extensible model for the web with powerful reasoning ability. Therefore, we foresee its popular usage in the near future.

Bibliography

[1] *Trusted Computer System Evaluation Criteria (Orange Book)*. United States Department of Defense, DoD Standard 5200.28-STD., December 1985. 16, 17

[2] Martín Abadi, Michael Burrows, Butler Lampson, and Gordon Plotkin. A calculus for access control in distributed systems. *ACM Transactions on Programming Language and Systems*, 15(4):706–734, 1993. 29, 30

[3] F. Baader, C. Lutz, and B. Suntisrivaraporn. CEL—a polynomial-time reasoner for life science ontologies. In U. Furbach and N. Shankar, editors, *Proceedings of the 3rd International Joint Conference on Automated Reasoning (IJCAR'06)*, volume 4130 of *Lecture Notes in Artificial Intelligence*, pages 287–291. Springer-Verlag, 2006. 106

[4] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The description logic handbook: theory, implementation, and applications*. Cambridge University Press, New York, NY, USA, 2003. 2, 31, 42, 54, 57, 60, 122, 143

[5] Albert-László Barabási. *Linked : the new science of networks*. Perseus Pub., Cambridge, Mass., 2002. 88

[6] Steve Barker, Michael Leuschel, and Mauricio Varea. Efficient and flexible access control via jones-optimal logic program specialisation. *Higher-Order and Symbolic Computation*, 2007. 106

[7] Bebo. http://www.bebo.com/. 11

[8] Elisa Bertino, Piero Andrea Bonatti, and Elena Ferrari. Trbac: A temporal role-based access control model. *ACM Transactions on Information System Security*, 4(3):191–233, 2001. 16, 25

[9] Elisa Bertino, Barbara Catania, Elena Ferrari, and Paolo Perlasca. A logical framework for reasoning about access control models. *ACM Transactions on Information and System Security*, 6(1):71–127, 2003. 29, 31, 106

[10] B.Lampson. Protection. In *Proc. 5th Princeton Conf. on Information Sciences and Systems, Princeton, 1971. Reprinted in ACM Operating Systems Rev. 8, 1*, pages 18–24, 1971. 15, 17, 37, 75

[11] Blogger. http://www.blogger.com/. 1, 10

[12] Paolo Bouquet, Fausto Giunchiglia, Frank Harmelen, Luciano Serafini, and Heiner Stuckenschmidt. C-owl: Contextualizing ontologies. In *ISWC*, pages 164–179, 2003. 102

[13] Krysia Broda and Alessandra Russo. Compiled Labelled Deductive Systems for Access Control (2005). In *We Will Show Them: Essays in Honour of Dov Gabbay*, volume 1, pages 309–338. College Publications, 2005. 29

[14] Krysia Broda and Alessandra Russo. Compiled Labelled Deductive Systems for Access Control (2005). In *We Will Show Them: Essays in Honour of Dov Gabbay*, volume 1, pages 309–338. College Publications, 2005. 30

[15] Ji-Won Byun, Elisa Bertino, and Ninghui Li. Purpose based access control of complex data for privacy protection. In *SACMAT '05: Proceedings of the tenth ACM symposium on Access control models and technologies*, pages 102–110, New York, NY, USA, 2005. ACM. 16, 81, 83

[16] Ji-Won Byun and Ninghui Li. Purpose based access control for privacy protection in relational database systems. *VLDB J.*, 17(4):603–619, 2008. 25

[17] Jung-Hwa Chae and Nematollaah Shiri. Formalization of rbac policy with object class hierarchy. In Ed Dawson and Duncan S. Wong, editors, *ISPEC*, volume 4464 of *Lecture Notes in Computer Science*, pages 162–176. Springer, 2007. 24, 29, 31

[18] Junghwa Chae. Towards modal logic formalization of role-based access control with object classes. In *FORTE*, pages 97–111, 2007. 29

[19] Peter P. Chen. The entity-relationship model - toward a unified view of data. *ACM Transactions on Database Systems*, 1(1):9–36, 1976. 2, 28, 36

[20] M. Coetzee and J. H. P. Eloff. Towards web service access control. *Computers & Security*, 23(7):559–570, October 2004. 106

[21] Matthew Collinson and David Pym. Algebra and logic for access control. Technical Report HPL-2008-75, July 2008. Submitted to Formal Aspects of Computing. 29, 30

[22] Dmoz. http://www.dmoz.org/. 8

[23] F. Dridi, B. Muschall, and G. Pernul. Administration of an rbac system. *System Sciences, 2004. Proceedings of the 37th Annual Hawaii International Conference on*, pages 6 pp.–, Jan. 2004. 23

[24] eBay, Inc. Web site, 1995. http://www.ebay.com/. 1, 25

[25] OASIS eXtensible Access Control Markup Language (XACML) TC. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml. 28

[26] E.Zolin. Dl-navigator. http://www.cs.man.ac.uk/ ezolin/dl/. 124

[27] Facebook. http://www.facebook.com/. 1, 10

[28] D. Ferraiolo and R. Kuhn. Role-based access controls. In *15th NIST-NCSC National Computer Security Conference*, pages 554–563, 1992. 20

[29] David F. Ferraiolo, Ravi S. Sandhu, Serban I. Gavrila, D. Richard Kuhn, and Ramaswamy Chandramouli. Proposed NIST standard for role-based access control. *Information and System Security*, 4(3):224–274, 2001. 20, 21, 23, 38, 49, 67, 69, 77, 78, 93

[30] T. Finin, A. Joshi, L. Kagal, J. Niu, R. Sandhu, W. Winsborough, and B. Thuraisingham. Rowlbac: representing role based access control in owl. In *SACMAT '08: Proceedings of the 13th ACM symposium on Access control models and technologies*, pages 73–82, New York, NY, USA, 2008. ACM. 31, 32

[31] Flickr. http://www.flickr.com. 1, 10

[32] OASIS Advancing Open Standards for the Information Society. http://www.oasis-open.org/. 28

[33] Friendster. http://www.friendster.com/. 11

[34] Serban I. Gavrila and John F. Barkley. Formal specification for role based access control user/role and role/role relationship management.

In *RBAC '98: Proceedings of the third ACM workshop on Role-based access control*, pages 81–90, New York, NY, USA, 1998. ACM. 55

[35] Fausto Giunchiglia, Maurizio Marchese, and Ilya Zaihrayeu. Towards a theory of formal classification. In *CandO 2005,AAAI-05*, Pittsburgh, Pennsylvania, USA, July 9-13 2005 2005. 89, 96

[36] Fausto Giunchiglia, Maurizio Marchese, and Ilya Zaihrayeu. Encoding classifications into lightweight ontologies. In *ESWC*, pages 80–94, 2006. 92, 96

[37] Fausto Giunchiglia, Maurizio Marchese, and Ilya Zaihrayeu. Encoding classifications into lightweight ontologies. *J. Data Semantics*, 8:57–81, 2007. 92

[38] Fausto Giunchiglia, Pavel Shvaiko, and Mikalai Yatskevich. Semantic schema matching. In *In Proceedings of CoopIS*, pages 347–365. Springer, 2005. 97, 99

[39] Fausto Giunchiglia and Mikalai Yatskevich. Element level semantic matching. In *Meaning Coordination and Negotiation workshop at ISWC'04*. Hiroshima, Japan, November 2004. 97, 99

[40] Fausto Giunchiglia, Mikalai Yatskevich, and Enrico Giunchiglia. Efficient semantic matching. In *Proceedings of the 2nd European semantic web conference (ESWC'05)*. LNCS, Springer Verlag, 2005. 97, 99

[41] Fausto Giunchiglia, Mikalai Yatskevich, and Pavio Shvaiko. Semantic matching: Algorithms and implementation. *Journal on Data Semantics*, pages 1–38, 2007. 2, 95, 97, 98

[42] Fausto Giunchiglia and Ilya Zaihrayeu. Lightweight ontologies, 2008. 87, 89, 96

[43] Fausto Giunchiglia, Ilya Zaihrayeu, and Uladzimir Kharkevich. Formalizing the get-specific document classification algorithm. In *In 11th European Conference on Research and Advanced Technology for Digital Libraries*. LNCS Springer Verlag, 2007. 91, 96

[44] Fausto Giunchiglia, Rui Zhang, and Bruno Crispo. Relbac: Relation based access control. In *SKG '08: Proceedings of the 2008 Fourth International Conference on Semantics, Knowledge and Grid*, pages 3–11, Washington, DC, USA, 2008. IEEE Computer Society. vi, 36, 95

[45] Fausto Giunchiglia, Rui Zhang, and Bruno Crispo. Ontology driven community access control. In *SPOT2009 - Trust and Privacy on the Social and Semantic Web*, to appear 2009. vi, 2, 94, 95

[46] Jennifer Golbeck and James Hendler. Reputation network analysis for email filtering. In *In Proc. of the Conference on Email and Anti-Spam (CEAS), Mountain View*, 2004. 26

[47] S. Greco, N. Leone, and P. Rullo. Complex: An object-oriented logic programming system. *IEEE Transactions on Knowledge and Data Engineering*, 4(4):344–359, 1992. 31

[48] Adam Greenfield. *Everyware: The Dawning Age of Ubiquitous Computing*. Peachpit Press, Berkeley, CA, USA, 2006. 9

[49] V. Haarslev and R. Möller. Racer: A core inference engine for the semantic web. In *Proceedings of the 2nd International Workshop on Evaluation of Ontology-based Tools (EON2003), located at the 2nd International Semantic Web Conference ISWC 2003, Sanibel Island, Florida, USA, October 20*, pages 27–36, 2003. 106

[50] V. Haarslev and R. Möller. Racer: A core inference engine for the semantic web. In *Proceedings of the 2nd International Workshop on Evaluation of Ontology-based Tools (EON2003), located at the 2nd International Semantic Web Conference ISWC 2003, Sanibel Island, Florida, USA, October 20*, pages 27–36, 2003. 106

[51] U. Hustadt and R. A. Schmidt. MSPASS: Modal reasoning by translation and first-order resolution. In R. Dyckhoff, editor, *Automated Reasoning with Analytic Tableaux and Related Methods, International Conference (TABLEAUX 2000)*, volume 1847 of *Lecture Notes in Artificial Intelligence*, pages 67–71. Springer, 2000. 106

[52] Ismail Khalil Ibrahim and Werner Winiwarter. Winiwarter: Content-based management of document access control. In *14th International Conference on Applications of Prolog (INAP), Prolog Association of*, pages 78–86, 2001. 106

[53] Audun Josang and Roslan Ismail. The beta reputation system. In *In Proceedings of the 15th Bled Electronic Commerce Conference*, 2002. 26

[54] James Joshi, Elisa Bertino, and Arif Ghafoor. An analysis of expressiveness and design issues for the generalized temporal role-based access control model. *IEEE Transactions on Dependable and Secure Computing*, 2(2):157–175, 2005. 25

[55] James Joshi, Elisa Bertino, Usman Latif, and Arif Ghafoor. A generalized temporal role-based access control model. *IEEE Transactions on Knowledge and Data Engineering*, 17(1):4–23, 2005. 16, 25

[56] Thumrongsak Kosiyatrakul, Susan Older, and Shiu-Kai Chin. A modal logic for role-based access control. *Computer Network Security*, pages 179–193, 2005. 29

[57] Michael Lemay, Omid Fatemieh, and Carl A. Gunter. Policymorph: interactive policy transformations for a logical attribute-based access control framework. In *SACMAT '07: Proceedings of the 12th ACM symposium on Access control models and technologies*, pages 205–214, New York, NY, USA, 2007. ACM Press. 106

[58] Ninghui Li and Mahesh V. Tripunitara. Security analysis in role-based access control. *ACM Transactions on Information System Security*, 9(4):391–420, 2006. 66, 70

[59] Ninghui Li, Mahesh V. Tripunitara, and Ziad Bizri. On mutually exclusive roles and separation-of-duty. *ACM Transactions on Information System Security*, 10(2):5, 2007. 68, 69

[60] LinkedIn. http://www.linkedin.com/. 10

[61] Fabio Massacci. Reasoning about security: A logic and a decision method for role-based access control. In *ECSQARU-FAPR*, pages 421–435, 1997. 29, 30

[62] W. McCune. Otter 3.3 reference manual and guide, 2003. 30

[63] George A. Miller. Wordnet: A lexical database for english. *Communications of the ACM*, 38:39–41, 1995. 98

[64] Boris Motik. Kaon2. http://kaon2.semanticweb.org/. 106

[65] Matthew J. Moyer and Mustaque Ahamad. Generalized role-based access control. In *ICDCS*, pages 391–398, 2001. 16, 24

[66] MySpace. http://www.myspace.com/. 10

[67] Friend of a Friend. http://www.foaf-project.org/. 26, 88

[68] Web of Trust. http://www.mywot.com/. 26, 88

[69] OWL. http://www.w3.org/TR/owl-guide/. 32, 87, 108

[70] OWL-API. http://owlapi.sourceforge.net/. 87

[71] OWL-DL. http://www.w3.org/TR/owl-semantics/. 87, 95

[72] Jaehong Park and Ravi Sandhu. The ucon¡sub¿abc¡/sub¿ usage control model. *ACM Transactions on Information and System Security*, 7(1):128–174, 2004. 16

[73] Raymond Reiter. On closed world data bases. In *Logic and Data Bases*, pages 55–76, 1977. 4, 106, 117, 121

[74] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, 1996. 15, 20, 28

[75] Leo Sauermann, Ansgar Bernardi, and Andreas Dengel. Overview and Outlook on the Semantic Desktop. In Stefan Decker, Jack Park, Dennis Quan, and Leo Sauermann, editors, *Proceedings of the 1st Workshop on The Semantic Desktop at the ISWC 2005 Conference*, volume 175 of *CEUR Workshop Proceedings*, pages pp. 1–19. CEUR-WS, November 2005. http:// CEUR-WS.org/Vol-175/. 38

[76] S. Shepler, B. Callaghan, D. Robinson, R. Thurlow, C. Beame, M. Eisler, and D. Noveck. Network file system (nfs) version 4 protocol, 2003. 18, 75

[77] E. Sirin, B. Parsia, B. Cuenca Grau, A. Kalyanpur, and Y. Katz. Pellet: A practical owl-dl reasoner. *Submitted for publication to Journal of Web Semantics.*, 2003. 105, 106, 117, 120

[78] SWRL. http://www.w3.org/Submission/SWRL/. 124

[79] D. Tsarkov and I. Horrocks. Fact++ description logic reasoner: System description. In *Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR 2006)*, volume 4130 of *Lecture Notes in Artificial Intelligence*, pages 292–297. Springer, 2006. 106, 117

[80] Jeffrey D. Ullman. *Principles of Database and Knowledge-Base Systems, Volume I.* Computer Science Press, 1988. 31

[81] Jeffrey D. Ullman. *Principles of Database and Knowledge-Base Systems, Volume II.* Computer Science Press, 1989. 31

[82] Brett Walker and Stuart J. Barnes. Wireless sales force automation: concept and cases. *IJMC*, 3(4):411–427, 2005. 60, 67, 121

[83] Wikipedia. http://www.wikipedia.org/. 1

[84] XML. http://www.w3.org/XML/. 28

[85] Yahoo! http://dir.yahoo.com/. 8

[86] Bin Yu and Munindar P. Singh. A social mechanism of reputation management in electronic communities. In *Cooperative Information Agents*, pages 154–165, 2000. 26

[87] Rui Zhang. Automatic access control rule generation via semantic matching. *Workshop on Matching and Meaning 2009 (WMM09)*, to appear 2009. vi

[88] Rui Zhang, Bruno Crispo, and Fausto Giunchiglia. Relbac:design and run time reasoning about web access control policies. In *International Symposium on Policies for Distributed Systems and Networks (POLICY 09)*, to appear 2009. vi, 93, 99

[89] Chen Zhao, NuerMaimaiti Heilili, Shengping Liu, and Zuoquan Lin. Representation and reasoning on rbac: A description logic approach.

In Dang Van Hung and Martin Wirsing, editors, *ICTAC*, volume 3722 of *Lecture Notes in Computer Science*, pages 381–393. Springer, 2005. 29, 31

Appendix A

Brief Introduction of Description Logic

Description logic (DL) is first introduced as a formalism for representing knowledge. Basic DL [4] contains basic attribute language ($\mathcal{AL}$) as in Table A.1. The most common extensions of $\mathcal{AL}$ are listed in Table A.2.

In DLs, a distinction is drawn between the so-called TBox (terminological box) and the ABox (assertional box). In general, the TBox contains sentences describing concept hierarchies (i.e., relations between concepts) while the ABox contains "ground" sentences stating where in the hierarchy individuals belong (i.e., relations between individuals and concepts)

The terminology axioms have the forms as the following:

$$C \sqsubseteq D(R \sqsubseteq Q) \mid \ (inclusion)$$
$$C \equiv D(R \equiv Q) \mid \ (equality)$$

Where C,D are concepts and R,S are roles. An interpretation I satisfies inclusion $C \sqsubseteq D$ if $C^I \subseteq D^I$ (I satisfies inclusion $R \sqsubseteq Q$ if $R^I \subseteq Q^I$), and it satisfies equality $C \equiv D$ if $C^I = D^I$ (I satisfies equality $R \equiv Q$ if $R^I = Q^I$). An equality with left-side an atomic concept is called a definition. A set of definitions T is a terminology (or TBox) if no symbolic name is defined no more than once, that is every atomic concept A can appear on left-side

Table A.1: Basic Description Logic-Attribute Language ($\mathcal{AL}$)

Constructor	**Syntax**	**Semantic**
Atomic concept	A	A^I
Top concept	$\top$	$\triangle^I$
Bottom concept	$\bot$	$\emptyset$
Complement of atomic concept	$\neg A$	$\triangle^I - A^I$
Intersection of atomic concepts	$C \sqcap D$	$C^I \cap D^I$
Universal value restriction	$\forall R.C$	$\{a \in \triangle^I \mid \forall b.R(a,b) \rightarrow C(b)\}$
Limited existential restriction	$\exists R.\top$	$\{a \in \triangle^I \mid \exists b.R(a,b)\}$

of an axiom at most once.

Here's an example (eg1) about a bank service: all the system users are divided into two groups the customers and the employees (intersection may exists). There are 3 positions among bank employees, the teller, the auditor and the manager. The teller serves the customer. The auditor audits the teller. We can model with the following TBox (TB1):

$$User \equiv Customer \sqcup Employee, Teller \equiv serve.Customer,$$
$$Auditor \equiv audit.Teller, Manager \equiv manage.Employee,$$
$$Teller \sqsubseteq Employee, Auditor \sqsubseteq Employee, Manager \sqsubseteq Employee.$$

So that in this terminology, we have the following result:

$$T \vDash Manager \sqsubseteq User \text{ or } Manager \sqsubseteq_T User$$

by which AL mean the proposition '$Manager \sqsubseteq User$' can be inferred from TBox 'TB1'.

The world description axiom (ABox) describes a specific state of affairs of an application domain in terms of concepts and roles. ABox has the following forms:

$$C(a),\ R(b,c)$$

in which a, b, c are individuals, C is a concept and R is a role. The two forms are called concept assertion and role assertion respectively. Inter-

Table A.2: Extended Description Logic

Constructor	**Syntax**	**Semantic**
Existential restriction	$\exists R.C$	$\{a \in \triangle^I \mid \exists b.R(a,b) \wedge C(b)\}$
Number restriction	$\leq nR$	$\{a \in \triangle^I \mid \ \mid \{b \mid R(a,b)\} \mid \leq n\}$
	$\geq nR$	$\{a \in \triangle^I \mid \ \mid \{b \mid R(a,b)\} \mid \geq n\}$

pretations of the ABox are former interpretations of concepts and roles in addition to mappings from individual name to an element in the interpretation function domain, such as to map a to $a^I \in \triangle^I$. For convenience AL introduces individual name with the constructor ‘set’ and ‘fill’.

$$\{a_1, ..., a_n\} \mid (set)$$
$$(R : a) \mid (fill)$$

The interpretations of these constructors are: $\{a_1, ..., a_n\}^I = \{a_1^I, ..., a_n^I\}$ and $(R : a)^I = \{b \in \triangle^I \mid (b, a^I) \in R^I\}$.