A SEMANTIC BASED POLICY MANAGEMENT FRAMEWORK FOR CLOUD COMPUTING ENVIRONMENTS

by

Hassan Takabi

B.Sc., Computer Engineering (Software Engineering), AmirKabir University of Technology, 2004

M.Sc., Information Technology (Computer Networks), Sharif University of Technology, 2007

Submitted to the Graduate Faculty of

the School of Information Sciences in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

University of Pittsburgh

2013

UNIVERSITY OF PITTSBURGH

SCHOOL OF INFORMATION SCIENCES

This dissertation was presented

by

Hassan Takabi

It was defended on

July 12, 2013

and approved by

James B. D. Joshi, Associate Professor, School of Information Sciences

Hassan A. Karimi, Professor, School of Information Sciences

Michael B. Spring, Associate Professor, School of Information Sciences

Prashant Krishnamurthy, Associate Professor, School of Information Sciences

Gail-Joon Ahn, Associate Professor, Arizona State University

Dissertation Director: James B. D. Joshi, Associate Professor, School of Information Sciences

A SEMANTIC BASED POLICY MANAGEMENT FRAMEWORK FOR CLOUD COMPUTING ENVIRONMENTS

Hassan Takabi, PhD

University of Pittsburgh, 2013

Cloud computing paradigm has gained tremendous momentum and generated intensive interest. Although security issues are delaying its fast adoption, cloud computing is an unstoppable force and we need to provide security mechanisms to ensure its secure adoption.

In this dissertation, we mainly focus on issues related to policy management and access control in the cloud. Currently, users have to use diverse access control mechanisms to protect their data when stored on the cloud service providers (CSPs). Access control policies may be specified in different policy languages and heterogeneity of access policies pose significant problems.An ideal policy management system should be able to work with all data regardless of where they are stored. Semantic Web technologies when used for policy management, can help address the crucial issues of interoperability of heterogeneous CSPs.

In this dissertation, we propose a semantic based policy management framework for cloud computing environments which consists of two main components, namely policy management and specification component and policy evolution component. In the policy management and specification component, we first introduce policy management as a service (PMaaS), a cloud based policy management framework that give cloud users a unified control point for specifying authorization policies, regardless of where the data is stored. Then, we present semantic based policy management framework which enables users to specify access control policies using semantic web technologies and helps address heterogeneity issues of cloud computing environments. We also model temporal constraints and restrictions in GTRBAC using OWL and show how ontologies can be used to specify temporal constraints. We present

a proof of concept implementation of the proposed framework and provide some performance evaluation.

In the policy evolution component, we propose to use role mining techniques to deal with policy evolution issues and present StateMiner, a heuristic algorithm to find an RBAC state as close as possible to both the deployed RBAC state and the optimal state. We also implement the proposed algorithm and perform some experiments to demonstrate its effectiveness.

Keywords: cloud computing, policy management, semantic web, access control, policy evolution, role mining.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

PREFACE

I am grateful to Professor James B. D. Joshi, my advisor, for giving me the opportunity to work under his supervision and work as a member of LERSAIS lab.

I would also like to thank my dissertation committee members, Professor Hassan A. Karimi, Professor Michael B. Spring, Professor Prashant Krishnamurthy and Professor Gail-Joon Ahn for their valuable feedbacks on my work.

I also thank my colleagues at the LERSAIS lab, Amirreza Masoumzadeh Tork, Saman Taghavi Zargar, Lei Jin, Nathalie Angel Baracaldo, Xulian Long, Jesus Gonzales, and Yue Zhang for the fruitful discussions and the knowledge they shared with me.

I also acknowledge that the research presented in this dissertation has been supported by the US National Science Foundation Intelligent Information Systems (IIS) CAREER award IIS-0545912.

1.0 INTRODUCTION

Cloud computing paradigm has recently generated intensive interest within research communities in both academia and industry. It separates data resources from the underlying infrastructure and the approaches used to deliver it. Cloud computing essentially aims to consolidate the economic utility model with the evolutionary development of existing computing technologies such as distributed services, applications, information and infrastructure consisting of pools of computers, networks, information and storage resources [28]. Cloud computing is a very important paradigm that promises to provide significant cost reduction through optimization and the increased operating and economic efficiencies in computing [29]. It has shown tremendous potential to enhance collaboration, agility, scale, and availability [28].

Despite the enormous opportunity and value that the cloud presents for organizations, several surveys of potential cloud adopters indicate that security and privacy challenges are the number one concern hindering its adoption and it will continue to keep some companies from embracing cloud computing [3]. However, cloud computing appears to be an unstoppable force because of its potential benefits. Hence, understanding the security and privacy risks in cloud computing and developing effective solutions are critical to the success of this new computing paradigm [29].

In this dissertation, we propose solutions to policy management and specification and policy evolution issues in cloud computing environments.

1.1 PROBLEM STATEMENT

In this section, we discuss our problem statement and briefly present challenges which are the focus of this dissertation. More in depth discussion on the motivation, the challenges and the proposed research tasks will be presented in chapter 2.

Policy Management and Specification: The cloud computing environments do not allow use of a single authorization mechanism, single policy language or single management tool for users who use various CSPs. Each CSP has its own access control mechanism that typically has limited capability to support flexible user's fine-grained security and privacy requirements [29]. This approach is not user/customer centric and hence, can be a significant barrier to its widespread adoption. The specification of access control policies clearly is the responsibility of the organization/users deploying the cloud service. However, not only do the CSPs dictate how these policies should be specified but each also does it in its own way [28]. In an ideal access control scheme, users should be able to manage policies to govern access to their information and resources from a central location regardless of where they are stored. Having a centralized policy management helps users to use same access policies in multiple CSPs.

Currently, users must use diverse access control mechanisms to secure their data stored at different CSPs. Access control policies may be specified in diverse policy languages and maintained separately at every CSP. When such diverse mechanisms are used, they add considerable overhead, especially since they often lack flexibility with respect to functionality as well as user interfaces [38]. This may frustrate users and make them feel that they have no control on where their data ends up and how it is used. The challenge here is to design an access control system that can be used across services from different providers.

Security solutions delivered as cloud-based services will have a dramatic impact on the industry. Cloud computing will enable security controls and functions to be delivered in new ways and by new types of service providers. It will also enable customers to use security technologies and techniques that are not otherwise cost-effective. Enterprises that use cloud-based security services to reduce the cost of security controls and address the new security challenges that cloud based computing will bring are most likely to prosper.

Heterogeneity of access policies raises significant problems in managing them in cloud computing environments [28]. Semantic Web technologies when used for policy management, can provide runtime extensibility, adaptability and ability to analyze policies at different levels of abstraction. Hence, it can help address the crucial issues of heterogenous policies at CSPs [34]. The Semantic Web is an extension of the World Wide Web that allows knowledge, information and data to be shared on the Web and reused across applications and enterprises [26].

Web Ontology Language (OWL) is a standard knowledge representation and specification language for the Semantic Web, and hence it is a promising technology for addressing access control issues in cloud computing environments [47]. In order to specify a policy, one should be able to precisely specify classes of subjects, objects, actions, etc. Using OWL gives us a natural and efficient way of specifying these classes and access policies. The second advantage is that OWL is based on description logic and we can translate access policies expressed in OWL to other policy languages and formalisms [34].

Policy Evolution: The policies that are defined to protect resources stored on the cloud, will evolve over time. We assume that RBAC is access control mechanism of choice in our proposed system and it needs to be updated to meet the changes and handle policy evolutions. The idea of role mining can be used as a promising approach to deal with this issue. However, rather than adopting a completely different RBAC system and redefining everything from scratch, we present an algorithm to find an RBAC state as similar as possible to both the existing state and the optimal state.

1.2 PROPOSED RESEARCH AND CONTRIBUTIONS

Towards addressing the above mentioned challenges, the goal of this dissertation is to propose a semantic based policy management framework for cloud computing environments that is able to allow the users to specify access policies that are applied to their resources using a unified framework regardless of where the resources are stored. We also propose to use role mining to address the policy evolution issue. In particular, the research presented in this

dissertation makes the following contributions:

- We propose Policy Management as a Service (PMaaS), a cloud based policy management framework that puts users in control of their resources which may be scattered across multiple CSPs. It is designed to give cloud users a unified control point for specifying authorization policies, no matter where all the data is stored and distributed on the cloud. It enables users to manage access policies using a centralized policy manager which provides capabilities for specifying access policies.
- We present lessons we have learned from a case study where we developed of a unified policy management system for some real world cloud services.
- We introduce a semantic based policy management framework for cloud computing environments as an instance of the previously proposed PMaaS framework. Our proposed semantic based policy management framework is designed to give cloud customers a unified control point for specifying authorization policies and enables users to specify, edit and manage access policies using a centralized policy manager which uses semantic web technologies for specifying access policies and a basis for addressing heterogeneity issue of cloud computing environments.
- We define OWL ontologies to represent all entities involved in access control policy specification in cloud computing environments, address different collaboration scenarios among CSPs, and represent temporal constraints and restrictions in GTRBAC to support the specification of time based access needs.
- We present a proof of concept implementation of the proposed semantic based policy management framework and provide some performance evaluation.
- We propose to use role mining techniques to address policy evolution issues. We formally define the problem of mining role hierarchy with minimal perturbation and present StateMiner, presents a heuristic solution to find an RBAC state as close as possible to both the deployed RBAC state and the optimal state.
- We present a proof of concept implementation of StateMiner and provide experimental results to show its effectiveness.

1.3 ORGANIZATION

The rest of this dissertation is organized as follows. In chapter 2, we present two use case scenarios to motivate the need for our proposed framework and lessons we learned from a case study where we developed a unified policy management system for multiple CSPs. Then, we discuss and identify the challenges which are the focus of this dissertation and present the research tasks carried out in this dissertation. In chapter 3, we present relevant background and related work on policy management and role engineering. In chapter 4, we present our proposed semantic based policy management framework for cloud computing environments which includes two main components that correspond to the two proposed tasks. We explain each of these components in detail. Finally, we conclude the dissertation and discuss future work in chapter 5.

2.0 CHALLENGES AND PROPOSED RESEARCH

The cloud computing environment can be deemed as an instance of the multi-domain environment where each domain employs different security, privacy and trust requirements and potentially employ various mechanisms, interfaces, and semantics. Such domains could represent individually enabled services or other infrastructural or application components. In our previous work, we have presented security and privacy challenges that cloud computing raises and how they are related to various delivery and deployment models, and are exacerbated by the unique aspects of cloud [28]. We have also introduced a comprehensive security framework for cloud computing environments, discussed existing solutions and proposed some approaches to deal with security and privacy challenges [29]. In this dissertation, we focus on various aspects of policy management and access control issues in the cloud computing environments. We also address policy evolution issues in the cloud.

2.1 POLICY MANAGEMENT IN CLOUD COMPUTING ENVIRONMENTS

Heterogeneity and diversity of services, and the domains' diverse access requirements in cloud computing environments would require fine-grained access control policies. In particular, access control services should be flexible enough to capture dynamic, context or attribute/credential based access requirements, and facilitate enforcement of the principle of least privilege. Such access control services may need to integrate privacy protection requirements expressed through complex rules. It is important that the access control system employed in clouds is easily managed and its privilege distribution is administered efficiently.

It is important to ensure that the cloud delivery models provide generic access control interfaces for proper interoperability, which demands for a policy neutral access control specification and enforcement framework that can be used to address cross-domain access issues [20].

Role Based Access Control (RBAC) has been widely accepted as the most promising access control model because of its simplicity, flexibility in capturing dynamic requirements, and support for the principle of least privilege and efficient privilege management [11, 20]. Furthermore, RBAC is policy neutral, can capture a wide variety of policy requirements, and is best suited for policy integration needs discussed earlier. Recent RBAC extensions such as credential-based RBAC [13], Generalized Temporal RBAC (GTRBAC) [11], and location based RBAC models [17] provide necessary modeling constructs and capabilities to capture context based fine-grained access control requirements.

The cloud computing environments do not allow use of a single authorization mechanism, single policy language or single management tool for users who use various CSPs. Each CSP has its own access control mechanism that typically has limited capability to support user's security requirements [29]. The specification of access control policies clearly is the responsibility of the organization/users using the cloud service to store their resources. However, not only do the CSPs dictate how these policies should be specified but each also does it in its own way [28]. An ideal access control scheme must be able to work with all data regardless of where they are stored. Users should be able to manage policies to govern access to their information and resources from a central location.

In the following, we present two use case scenarios to motivate the need for the proposed unified policy management framework of a cloud based policy management service and then discuss how individual users and organizations that use cloud services can benefit from it. We also present lessons we learned from a case study where we developed a unified policy management system for multiple real world cloud services.

2.1.1 Use Case Scenarios

In first scenario, consider Alice is a PhD student and uses different applications and services for various purposes. She is working on a project and wants to have access to the project files from anywhere. Sometimes she works at her office using her PC and sometimes she works at home or a coffee shop using her laptop. In order to properly synchronize the project files, she uses *Dropbox*, a file hosting service which uses cloud computing to enable users to store and share files and folders with others across the Internet using file synchronization (http://www.dropbox.com). Since she collaborates with some other researchers on the project, sometimes she needs to share some of the files she stores at *Dropbox* with her colleagues.

Alice also has other documents and spreadsheets that include important content like financial data; she uses *Google Docs*, a service to create and share various types of files online and access them from anywhere (http://docs.google.com). Moreover, she stores some of her older files at *Amazon S3*, an online storage service that provides unlimited storage through a simple web services interface (http://aws.amazon.com/s3). She occasionally shares some of these files with family members. Furthermore, she uses *Mint*, an online personal finance service, to manage her financial and budget planning (http://www.mint.com). Sometimes she may want to share some of these files with a family member or a close friend.

Some of the above mentioned applications are used for convenience and others for sharing and collaboration purposes. With the increasing amount of data that Alice puts online, managing access control for her resources becomes difficult and time consuming. Each of these applications has its own access policy mechanism forcing users to specify separate access control policies for each application. However, with resources scattered across multiple applications, it is difficult to manage access to them. Alice needs to understand the applications' policy mechanisms and specify access policies in their specific policy languages which is a challenging task for her and most users like her. These mechanisms often are either very simple and inflexible or complicated and difficult to use.

Moreover, introducing new access rules or modifying existing ones is problematic due to heterogeneity of these access policy mechanisms. Suppose Alice wants to modify an access

rule to a set of resources, or a new colleague is added to her current project and she needs to share some resources with him. In order to do this, she needs to scan all her applications and services to modify access policy rules.

In the second scenario, consider an example where patients use multiple medical providers. For instance, $MedicalProvider_1$ is used for general medicine, $MedicalProvider_2$ for dental needs, $MedicalProvider_3$ is a heart hospital and $InsuranceProvider$ provides insurance. Each of these medical providers use one or more CSPs for various purposes. For example, one CSP provides email service, another CSP is used for billing purpose, the third one is used for appointment scheduling, and another one provides medical records management service.

Patients' information is stored on various CSPs and they want to protect their information from unauthorized users. For example, only cashiers need to have access to billing information which is stored on the CSP that is used for billing purpose, receptionists need access to information related to appointment scheduling while doctors and nurses need to have access to medical records management.

When it comes to access control and policy management, patients may have same information with multiple medical providers which are stored on their associated CSPs. Also, patients may want to share results of a specific test which is managed by $MedicalProvider_1$ and stored on CSP_1 with some users of $MedicalProvider_3$. There are also situations where $MedicalProvider_1$ and $MedicalProvider_3$ may want to share some customer information to have a larger database and be able to obtain more useful information and provide better service.

From the above two scenarios, we can see some of the key issues: applications and services have their own access control mechanisms, users need to understand the applications' access control mechanisms, and are forced to specify separate access control policies for each application using different policy languages. A centralized policy management service could help users to better manage security and provide them with a better view on access control policies applied to the their resources on different cloud services. Using the proposed framework, the access control policies can be applied to a distributed set of resources hosted on various CSPs. It gathers information from all of the services users use and provides them with an interface to centrally manage access to their resources regardless of where they are

stored at and what CSP they belong to.

2.1.2 Case Study and Lessons Learned

We analyze the scenarios described previously and discuss our case study implementation in order to identify limitations of existing access control mechanisms for the cloud and determine requirements that a policy management framework should have, to be able to address those limitations and be an appropriate candidate for the cloud computing environments.

Considering the aforementioned scenarios, we investigated access control mechanisms of some of the existing cloud services with the goal of developing a small scale unified policy management for cloud computing environments to enable users to manage access to their resources in a central location no matter where the resources are stored in the cloud. We analyzed more than 20 services looking at various features for each, including the following:

- Authentication mechanism to determine whether it is identity based, email based or some other mechanism.
- How users can share resources with other users
- What privacy/access setting options it provides
- What policy language and mechanism it uses.
- What APIs it provides.
- Whether it allows users to change privacy settings using an API or in some other ways.
- Whether we can discover users' resources stored in the service.
- Whether it supports XACML [62] or similar technologies.

Based on information gathered from this analysis, we picked the following five service providers for our purpose: *Amazon S3*, *Dropbox*, *LinkedIn*, *Flickr* (http://www.flickr.com), and *Twitter* (http://www.twitter.com/). Here is a summary of how each service deals with policy management.

- In *Amazon*, resources are represented by either bucket or object. An object is any data item stored in the system and a bucket is a top-level container in which objects are stored; for example, files are represented by objects and folders using buckets. In order to manage *S3* accounts, one should use Amazon Web Services (AWS) login credentials

that are stored in an *AWSCredentials* object. *Amazon S3* uses access control lists (ACLs) to control who has access to buckets and objects in *S3*. By default, any bucket or object a user creates belongs to him and is not accessible to anyone else. The user can use JetS3t (a toolkit for Java programmers with an API for interacting with storage services) for access control lists to make buckets or objects publicly accessible, or to allow other *S3* members to access or manage objects.

- In *Dropbox*, one needs an app key, secret, username, and password to get a token for authentication. Function *getAccountInfo* is used to get an authentication token. It uses the key and secret to get the token, and then logs into the account using the username and password. A recursive function *getResource* can be used to get information about all the resources. Users can invite other users to share specific folders. Function *sharefile* is used for posting files to the *Dropbox* server while function *deletefile* is used for removing a file from the server. Function *getfile* is used for copying a file from server to the local machine. So, the sharing process could be done by copying the target file, deleting it from original folder, and then posting it to the public folder or a specific shared folder.
- In *LinkedIn*, we need an app key, secret, token key and token secret for authentication. Function *getProfile* is used to login and get all the information of the full profile from the user's account. Function *permissionGrant* is used to send an invitation to another user with his email address, first name and last name.
- In *Flickr*, an app key and a secret are required to get a token for authentication. Function *showPeople* is used to get the photos of the user. Function *getPermissions* determines whether photos are public, shared or private. Function *publicPhoto* can be used for changing the ACL to public while function *privatePhoto* can be used to change the ACL to private.
- *Twitter* offers two discrete REST APIs [46]. The REST API methods allow developers to access core Twitter data. This includes update timelines, status data, and user information. The Search API methods give developers methods to interact with the *Twitter* Search function. In order to use the *Twitter* API, one first has to register a client application that will be provisioned a consumer key and a secret. This key and secret scheme is similar to the public and private keys used in protocols such as SSH

[56]. Authentication can be achieved using The OAuth and the consumer key and secret will be used, in conjunction with an OAuth library to sign every request you make to the API. The APIs allow developers to work with followers, friends, tweets, etc.

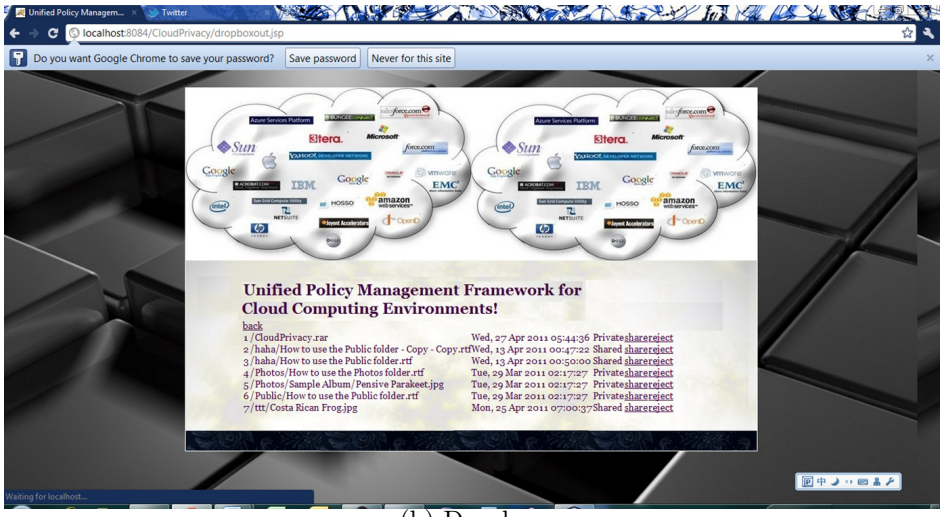
Figure 1 shows some screen shots of services and how users can modify their access control policy settings using the APIs they offer. Figures 1(a), 1(b) and 1(c) show resources and access control permissions related to *Amazon S3*, *Dropbox* and *Flickr*, respectively.

We developed a unified framework to support policy management for all these services in one central place. The goal was to provide users with a unified way of specifying access policies and then export policies into the services on behalf of the users. It includes various components such as an interface that provides users with a means to register applications and services they use, discover the resources each user has on various application services, specify policies and a component to export the policies into the services through their APIs. The framework was developed to do the following tasks:

- Get the name of service (i.e. *Dropbox*) with the username (i.e. Alice)
- Connect to the service
- Retrieve resources related to that username and display them
- Provide appropriate interfaces to the user to specify access policies/privacy settings
- Once the user specifies/ modifies access policies, it updates access control policies of the user in the associated services.

In order to integrate all five services, we used *servlet* and *jsp* to build a browser/server system so the users can access the framework using a browser. A servlet is the Java platform technology used for extending and enhancing the capabilities of Web servers where host applications accessed via a request-response programming model. It provides a component-based, platform-independent method for building Web-based applications, without the performance limitations of CGI programs. Although servlets can respond to any type of request, they are commonly used to extend the applications hosted by Web servers. Java Server Pages (JSP) is a Java technology that helps software developers serve dynamically generated web pages based on HTML, XML, or other document types. To deploy and run a Servlet, a Web container is required and is responsible for managing the life cycle of servlets, mapping

(a) Amazon



(b) Dropbox

(c) Flickr

Figure 1: The Screen Shots of Various Cloud Services

a URL to a particular servlet and ensuring that the URL requester has the correct access rights. A compatible web server with servlet container is also required to deploy and run a JSP.

The architecture is designed using model view controller (MVC). The *servlet* listens to the *jsp* page calls, runs the source code and redirects to the target page with the results inside the HTTP package. Based on the architecture, we developed a framework to integrate those five services. It displays all of users' resources associated with service and their access permissions in one central place. It also provides users with the ability to modify the permissions if they wish. Note that in some cases users' ability to specify/ modify permissions depends on expressive power of the access control mechanism that each CSP uses. For example, in *Amazon S3*), users can only share a file with another user but not a group.

Based on what we have learned from our attempt and findings of the case study, here we discuss the limitations of the existing policy management mechanisms and requirements for an appropriate policy management solution that addresses those limitations and challenges of the cloud environments.

Authorization mechanisms in the existing services are bound to service providers; each CSP employs its own authorization mechanism that is bound to its services and applications. In some cases, these access control mechanisms can only address simple scenarios. This limits the configuration of the application and it cannot be easily adapted to particular user's access control requirements. Alice, for example, has to use the solutions provided by *Dropbox*, *LinkedIn*, *GoogleDocs*, *Amazon S3*, and *Mint* which may not necessarily meet all her requirements. These solutions may not allow Alice to group users and assign permissions to such groups or may not support fine-grained access control policy rules. Most security novice users may choose their preferred services based on functionalities rather than security features. However, more security conscious users may decide to leave CSPs that do not support particular security features. However, we believe that users should be able to set their own access control policies for resources using their preferred policy management system. Moreover, the policy management system should enable users to apply a single access control policy across a set of distributed resources. For example, it should be possible to specify a policy that offers access only to Alice's family members, and attach that policy to a document

residing in *GoogleDocs*, and another document in *Mint*.

Cloud services are controlled by different authorities and often use different policy specification mechanisms. This leads to access control policies that are composed using diverse and potentially incompatible policy languages. For example, *Dropbox* uses a different access control model than *LinkedIn* that supports a more expressive and flexible access control language. Therefore, Alice is not able to specify access control policies once and apply these access control rules to her various resources such as photos, documents and video clips which may be spread across various cloud services. Moreover, if Alice decides to move some of her data from one CSP to another, for example from *Dropbox* to *Amazon S3*, then she may need to specify the policies again. Different CSPs may deploy different access control mechanisms and may force users to use their specific policy management tools. This could result in an inconsistent and inconvenient user experience. For example, Alice, to be able to share her resources efficiently and securely, must learn how to use interfaces and management tools at all her cloud services which may differ significantly from one service provider to another. We believe, in an ideal policy management system, users should be able to specify access control policies using a single policy management tool without being required to learn how to use each application's tool separately.

Each CSP employs its own authorization mechanism and the access control policies in existing solutions are distributed and heterogeneous. As a result, interoperation among service providers is difficult as services may not understand each other's authorization mechanism. Furthermore, users cannot have a holistic view of the access control policies applied to their resources over the cloud. In our scenario, Alice does not have a consolidated view of the access control policies applied to her information and resources in *Dropbox*, *LinkedIn*, *Google Docs*, *Amazon S3*, and *Mint*. Users should be provided with a holistic view of the access control policies applied to all their resources stored at various CSPs in order to enable them to have a better understanding of the access policies and help them in managing access to distributed resources. Additionally, with the heterogeneity of access policies, in order to introduce new access policies or modify existing policies, users need to go over all CSPs and configure access control policies appropriately which makes it a tedious task.

Currently, users must use diverse access control mechanisms to secure their data stored

at different CSPs. Access control policies may be specified in incompatible policy languages and maintained separately at every CSP. When such diverse mechanisms are used, they add considerable overhead, especially since they often lack flexibility with respect to functionality as well as user interfaces [38]. This may frustrate users and make them feel that they have no control on where their data ends up and how it is used. The challenge here is to design an integrated policy management system that can be used across services from different providers. Heterogeneity and distribution of access policies pose significant problems in managing them in cloud computing environments [28]. Furthermore, traditional non-semantic-based access control approaches are inadequate for supporting interoperability for cloud computing environments [31].

2.2 POLICY EVOLUTION

The policy specification component of our proposed framework is responsible to keep all policies applied to users' resources on the CSPs. These policies that are defined to protect resources across CSPs will evolve during time. As we will see later, these policies are based on RBAC model and it is known that maintenance of an RBAC system becomes an important issue once the system is in place [89]. An RBAC system needs to be updated to meet the changes and new permission distribution requirements. For example, new users may be added to the system, new applications that need new permissions may be added and existing applications may be deleted. When the initial RBAC configuration becomes messy and inefficient as a result of being used for a long time, many changes and updates, it is not a good idea to adopt a completely different RBAC system and redefine the system from scratch. Moreover, changes to the existing role set may cause disruptions to the system and prevent it from proper functioning.

The idea of role engineering could be used to deal with the policy evolution issues. Role engineering is the process of identifying a set of roles that is complete, correct and efficient, and then assigning users and permissions to these roles [73]. There are two general approaches to accomplish the task of role engineering: the *top-down approach* and the *bottom-*

up approach. The *top-down approach* uses a detailed analysis of business processes: defines particular job functions, decomposes them into smaller units, and finally creates roles for these units by associating needed permissions. Because there are large numbers of business processes, users and permissions in an organization, and also as such a process is human-intensive, it is a rather difficult task and hence believed to be slow and expensive. In order to overcome this drawback, researchers have proposed to use data mining techniques to discover roles from existing data. This *bottom-up approach* utilizes the existing system configuration data (in particular the user permission assignments) to define roles. It first considers the existing users' permissions before RBAC is implemented, and aggregates them into roles. Such a bottom-up approach is called *role mining*.

There have been several attempts to propose bottom-up approaches to mining roles [79, 80, 81, 82, 83, 85, 86, 87, 89]. However, there is no formal notion of goodness of a produced role set. *Vaidya et al.* have formally defined the role mining problem using the notion of *minimality* [81]. Without considering semantic meanings, minimality may serve as a good approximation for discovering *descriptive* roles, but generally it is not a good measure for goodness of discovered roles. Furthermore, the existing role mining techniques do not consider the existing RBAC configuration and try to define everything from scratch. These approaches are not acceptable when there already is an RBAC system in place [89].

2.3 OVERVIEW OF RESEARCH TASKS

Considering all the aforementioned challenges, we propose a semantic based policy management framework for cloud computing environments that is able to allow the users to specify their policies that are applied to their resources using a unified framework regardless of where the resources are stored. We also propose an approach based on role mining to handle the policy evolution issue.

2.3.1 Task I: Policy Management and Specification

We first introduce policy management as a service (PMaaS), a cloud based policy management framework that puts users in control of their resources which may be scattered across multiple CSPs. Its goal is to provide capabilities for users to manage access policies using a centralized policy manager which provides interfaces for specifying access policies and exporting them to the CSPs on behalf of the user.

It is also necessary to address semantic heterogeneity among different service providers' policies since they may have different approaches to provide access control mechanisms [3, 22, 21]. Based on the motivating scenarios, we propose a semantic based policy management framework for cloud computing environments. Semantic Web technologies can help address the crucial issue of semantic heterogeneity across multiple providers in the cloud [34]. In a policy management system, access control rules are specified based on representations of concepts and policy rules. In order to deal with the heterogeneity of cloud computing environments, these representations should be flexible and generic to fulfil various cloud providers' modeling requirements. OWL is a promising technology for addressing access control issues in cloud computing environments. In order to specify a policy, one should be able to precisely specify classes of subjects, objects, actions, etc. We propose to use combination of SWRL and OWL 2 to express the authorization policies and deductive processes of the system which may be used in authorization decision making process.

Our proposed semantic based policy management framework enables users to control access to their resources that may be scattered across multiple CSPs. It enables users to specify, edit and manage access policies using a centralized policy manager which uses semantic web technologies for specifying access policies. Using semantic web technologies help in addressing semantic heterogeneity issue in cloud computing environments.

We also present how to specify temporal constraints and restrictions using OWL in the Generalized Temporal RBAC (GTRBAC) model. The GTRBAC model combines the key features of the RBAC model with a temporal framework to address situations where processes and functions may have limited time spans or periodic temporal durations, and it is useful for applications with inherent temporal semantics such as workflow-based systems.

We model temporal constraints and restrictions in OWL and define OWL ontologies that represent temporal constraints and restrictions in GTRBAC.

2.3.2 Task II: Policy Evolution

With regards to policy evolution, we use role mining techniques to address this challenge in our proposed policy management framework. Our premise is that migrating to a new set of roles from existing set of roles should cause as little disruption as possible. So, the goal is to look for a set of roles as close as possible to both the existing set of roles and the optimal set of roles.

We formally define the problem of mining role hierarchy with minimal perturbation and present a heuristic algorithm to find an RBAC state as similar as possible to the existing state and the optimal state. In order to achieve our goal, we use the theory of formal concept analysis [94], which has been shown to provide a strong theoretical foundation for role engineering [89]. We also introduce two different measures: structural complexity for optimality of an RBAC state, and similarity of sets of roles for minimal perturbation. Our proposed algorithm, *StateMiner*, presents a heuristic solution to find an RBAC state as close as possible to both the deployed RBAC state and the optimal state.

3.0 BACKGROUND AND RELATED WORK

The US National Institute of Standards and Technology (NIST) cloud efforts intend to promote the effective and secure use of the technology within government and industry by providing technical guidance and promoting standards. The NIST defines *cloud computing* as follows: "*Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model is composed of five essential characteristics, three service models, and four deployment models.*" [1]. The five key characteristics of cloud computing include *on-demand self-service*, *ubiquitous network access*, *location-independent resource pooling*, *rapid elasticity* and *measured service*, all of which are geared toward using clouds seamlessly and transparently [1]. The main three key cloud delivery models are *software as a service (SaaS)*, *platform as a service (PaaS)*, and *infrastructure as a service (IaaS)* [29].

IaaS provides a set of virtualized infrastructural components such as virtual machines and storage on which the customers can build and run applications. The most basic component is a virtual machine (VM) and the virtual operating system where the application will eventually reside. Issues such as trusting the virtual machine image, hardening hosts, and securing inter-host communication are critical areas in IaaS. In PaaS, the cloud providers enables the programming environments to access and utilize the additional application building blocks. These programming environments have a visible impact on the application architecture such as constraints on what services the application can request from an operating system. For example, a PaaS environment may limit access to well-defined parts of the file system, thus requiring a fine-grained authorization service. SaaS provides application

software enabled as on-demand-services. As clients acquire and use software components from different providers, composing them securely and ensuring that information handled by these composed services are well protected become important.

The cloud deployment models are public cloud, private cloud, community cloud, and hybrid cloud composed of multiple clouds [2]. Public cloud refers to an external or publicly available cloud environment that is accessible to multiple tenants, while private cloud is typically a tailored environment with dedicated virtualized resources for a particular organization. Similarly, community cloud is tailored for a particular group of customers. Hybrid cloud is combination of two or more of the previous cloud deployment models.

Cloud computing has become a very attractive paradigm, with the potential to significantly reduce costs through optimization and increased operating and economic efficiencies [2, 3]. The architectural features of the cloud allow users to achieve better operating costs and be very agile by facilitating fast acquisition of services and infrastructural resources as and when needed. However, these unique features also give rise to various security concerns [4, 29]. Several surveys of potential cloud adopters also indicate that security and privacy is the primary concern hindering its adoption [4, 3], yet cloud computing appears to be growing fast because of its potential benefits [28].

The Cloud Security Alliance is an effort to facilitate the mission to create and apply best practices to secure cloud computing [2]. Its report, "Security Guidance for Critical Areas of Focus in Cloud Computing", outlines areas of concern and guidance for organizations adopting cloud computing. The goal is to provide security practitioners with a comprehensive roadmap for being proactive in developing positive and secure relationships with cloud providers [2].

Jaeger et al. discuss security challenges in the cloud, foundation of future systems' security and key areas for cloud system improvement [6]. *Kandukuri et al.* present security issues that have to be included in service layer agreement (SLA) in cloud computing environment [10]. *Jensen et al.* provide an overview on technical security issues of the cloud [9]. They start with real-world examples of attacks performed on the Amazon EC2 service, then give an overview of existing and upcoming threats to the cloud. They also briefly discuss appropriate countermeasures to these threats, and further issues to be considered in

future research. *Gruschka et al.* present taxonomies and classification criteria for attacks on cloud computing based on the notion of attack surfaces of the cloud computing scenario participants and try to anticipate the classes of vulnerabilities that will arise from the cloud computing paradigm [7].

Chen et al. try to frame the full space of cloud security issues by examining contemporary and historical perspectives from industry, academia, government, and black hat community [5]. They argue that most of cloud computing security issues are not fundamentally new nor they are fundamentally intractable. However, they suggest that two issues of the complexities of multiparty trust and mutual auditability are to some degree new to the cloud and propose future research direction for these issues.

In this section, we review some of the existing work related to our proposed policy management framework including Semantic Web and its application in policy management, cryptographic approaches used for data protection in cloud computing environments and role mining techniques.

3.1 POLICY MANAGEMENT AND SEMANTIC WEB

The Semantic Web is an extension of the World Wide Web that allows knowledge, information and data to be shared on the Web and reused across applications and enterprises [26]. In the Semantic Web, ontologies are used to specify a domain of interest that consists of terms representing individuals, classes of individuals, properties, and axioms that assert constraints over them. It provides a structured vocabulary that describes concepts and relationships between them as well as a specification of the meaning of terms used in the vocabulary [26].

Different ontology languages provide different facilities [34]. The W3C standards for ontology languages are based on RDF. The Web Ontology Language (OWL 2) is a family of standard knowledge representation languages for the Semantic Web based on Description Logic (DL) with a representation in RDF [47]. Using a reasoner, we can check whether all of the statements and definitions in the ontology are mutually consistent [47]. There are

different kinds of OWL 2 ontologies that differ in terms of expressiveness and computational complexity. There is a tradeoff between expressiveness and efficient reasoning. The more expressive the language is, the more difficult and less efficient the reasoning is.

W3C's Web Ontology Working Group defines three different subsets of OWL 2 including OWL 2 EL, OWL 2 QL and OWL 2 RL [47]. The OWL 2 EL is in particular useful for ontologies that have large number of classes and/ or properties. It captures the expressive power of many such ontologies and performs basic reasoning problems in a polynomial time with respect to the ontology's size. OWL 2 QL is useful for scenarios that include very large number of instance data, and the most important reasoning task is to answer queries. OWL 2 QL can perform complete and sound conjunctive query answering in a time that is logspace with respect to the size of the data and it could be used for reasoning problems such as ontology consistency and class expression subsumption using polynomial time algorithms. These two subsets target particular applications that need high performance reasoning algorithms but limit their expressiveness. On the other hand, OWL 2 RL is able to provide scalable reasoning power without having to sacrifice too much expressiveness power [34]. OWL 2 RL reasoning systems can be implemented using rule-based reasoning engines and most of the problems like ontology consistency, class expression satisfiability, class expression subsumption, instance checking, and conjunctive query answering can be implemented in polynomial time with respect to the size of the ontology.

The OWL 2 RL provides variety of constructors to describe knowledge and define property semantics like *inverseOf*, *transitiveProperty* or *reflexiveProperty*; constructors to define object semantics like *equivalentClass*, *disjointWith* or *unionOf*. It also provides cardinality restrictions like *minCardinality*, *allValuesFrom* or *someValuesFrom* among other features [47]. Furthermore, its reasoning could be implemented using rule-based engines that provide good performance and scalability [47]. The Semantic Web Rule Language (SWRL) can be used to enrich the models defined using OWL 2. SWRL is used for representing rules on the Semantic Web and enables expressing conditional knowledge by extending OWL 2 [48]. SWRL is not decidable so we use the DL-Safe context [49] that is a syntactic restriction of SWRL and is decidable.

There have been some efforts to use Semantic Web and OWL as a representation language

for access control policies. *Yague et al.* have developed the semantic access control (SAC) model that applies Semantic Web technologies to the access control in open, heterogeneous and distributed systems [33]. The model makes use of different layers of metadata to take advantage of the semantics of different components relevant for access decision purposes. The SAC model helps to achieve semantic interoperation among different components of access control systems. The authors have developed semantic policy language (SPL) for specification of access policies as an application of the SAC model.

Marin Perez et al. propose a Grid middleware based on Semantic Web technologies to define, manage and enforce security policies in a Grid computing [34]. They use the Globus Toolkit which is an open source software toolkit and a reference implementation for Grid systems as base middleware. The authors use ontologies to provide a representation of the underlying information system and the resources. They define access policies using semantic-aware rules which enables the administrator to create higher-level definitions with more expressiveness. Their proposed architecture supports multiple authorization domains, deals with heterogeneity of Grid systems and enables organizations to use their own domain concepts without having knowledge about the rest of participant organizations. They also provide a proof of concept implementation and tested in Globus to show the feasibility of the solution.

Finin et al. have introduced ROWLBAC, a representation of RBAC in OWL [99, 109]. They propose two different approaches: one maps roles to classes and subclasses, and the other maps roles to values. In the first case roles are represented as class of users, and the role hierarchy relation is mapped to the subsumption relation in OWL. Then it maps SoD constraints to class disjointness constraints in OWL. The second approach is to map classes onto individuals and bind users to classes through the property *role*. It models constraints through specialized properties e.g. DSoD and SSoD. However, a standard DL reasoner can not detect constraint violations and we need to add rules to the ontology that degrades performance. *Knetchel et al.* have proposed an approach that uses OWL for reasoning about RBAC authorizations [104]. The model can support both roles and class hierarchies. However, it does not take into consideration SoD constraints. *Heilili et al* have defined users and roles as classes [108]. In order to handle negative authorizations, they define

two corresponding classes for each role, each permission or prohibition has corresponding classes for roles and users. In other words, for each permission, there is a class of roles that has that permission, and then a class of users that has that permission. It is similar for each prohibition. *Di et al.* have described another approach using OWL to specify the RBAC constraints [107]. Their approach models roles, users, permissions, and session as classes, with properties to relate users to roles and roles to permissions. They also define functional mappings between sessions and roles and specify constraints such as separation of duty constraints, prerequisite constraints and cardinality constraints with OWL. However, in order to specify separation of duty and other constraints rules must be added.

Kolovski et al. have developed a DL-based analysis tool for XACML policies [106]. Their proposed approach represents a mapping between Description Logics and XACML as well as reasoning methods to verify properties of XACML policies. *Kagal et al.* have proposed a general framework based on semantic web technologies that supports general purpose policy systems and is able to solve mismatches among different policy languages [105]. *Knechtel et al.* have proposed RBAC-CH, an extension of Hierarchical RBAC [110]. The authors extend Hierarchical RBAC by using a class hierarchy of the accessed objects and present a concept to implement this model in a DL knowledge base using an OWL ontology. The permissions of roles are defined on object classes and then users permissions to objects automatically derived by a reasoning service. *Cirio et al.* have proposed an access control system for context-aware environments designed using Semantic Web technologies, namely OWL and Description Logic [111]. They adopt the RBAC model and extend it with contextual attributes. The authors have developed a high level OWL ontology to express the elements of an RBAC system and also a domain-specific ontology to capture the features of a sample scenario. They use a DL reasoner to classify users and resources, and verify the consistency of the access control policies. To the best of our knowledge, there is no work on time based access control policies that we address in this dissertation.

Alcarez Calero et al. propose a multi-tenancy authorization system for cloud computing that is suitable for middleware service in the PaaS layer [36]. Their proposed authorization model supports multi-tenancy, RBAC, hRBAC, path-based object hierarchies, and federation. The authors also present an architecture for implementing the authorization model,

describe a proof of concept implementation and its performance evaluation.

Hu et al. present a new Semantic Access Control Policy Language (SACPL) in order to overcome to the limitations of traditional access control systems in the cloud computing environments [35]. They introduce Access Control Oriented Ontology System (ACOOS) as the semantic basis of SACPL that aims to solve the interoperability issue of distributed access control policies. The ACOOS is used to annotate some syntax elements of XACML, such as subject, object, action and attribute variables with semantic information. The authors also add some syntax elements such as priority and confidentiality.

Calero et al. propose a multi-tenancy authorization system for cloud computing that is suitable for middleware service in the PaaS layer [36]. Their proposed authorization model supports multi-tenancy, RBAC, hRBAC, path-based object hierarchies, and federation. The authors also present an architecture for implementing the authorization model, describe a proof of concept implementation and its performance evaluation.

3.2 POLICY EVOLUTION: ROLE ENGINEERING IN RBAC

RBAC is an authorization model in which access decisions are based on the roles that users hold within an organization. In RBAC, roles represent functions within a given organization and access permissions are associated with roles instead of users [27]. Users can activate a subset of the roles which they are members of and easily acquire all the required permissions.

In order to deploy an RBAC system, one requires to first identify a complete set of roles. This process, known as role engineering, has been identified as one of the costliest tasks in migrating to RBAC. *Coyne* [73] was the first to propose the role engineering problem and describe the concept of the top-down approach. A number of subsequent papers have focused on the top-down approach [75, 76, 77, 78]. The role mining problem was first proposed by *Kuhlmann et al.* [79]. They have proposed a clustering technique similar to the k-means. *Schlegelmilch et al.* have proposed the ORCA that discovers roles by clustering them on permissions [80]. It constructs a role hierarchy, but it does not allow overlapping roles, which is a significant drawback. *Vaidya et al.* have proposed RoleMiner that is a two-phase

algorithm based on subset enumeration [82]. It first generates a set of candidate roles and orders them. Then, it calculates a priority metric from the number of users who have all the permissions assigned to the role and the number of users who have a superset of the role's permissions, and then selects roles from the candidate roles based on this priority metric.

Vaidya et al. have formally defined the role mining problem as a matrix decomposition problem, which they refer to as the RMP (Role Mining problem) problem [81]. They show that the RMP problem is NP-complete and there is a close relationship between RMP and several existing data mining problems such as the minimal tiling problem and the discrete basis problem. Furthermore, they have proposed several variants of the basic-RMP. The δ-approximate RMP allows a limited amount of inexactness, which may result in less number of roles. The min-noise RMP allows administrators to specify the number of roles yet mine the best possible set of roles. All these techniques are limited to mining RBAC systems that do not have a role hierarchy. *Lu et al.* use binary integer programming to model the basic RMP and its variants [85].

Zhang et al. have presented a heuristic algorithm for role mining, which models an RBAC state as a graph [83]. The algorithm starts with an initial RBAC state and iteratively improves the system using pairs of roles such that merging or splitting the two roles will result in a graph with a lower cost. *Colantonio et al.* propose RBAM (Role-Based Association-rule Mining) to leverage the cost metric to find candidate role-sets with the lowest possible administration cost [84]. *Frank et al.* present a probabilistic model for role mining problem [91]. They show how roles can be inferred from data using a machine-learning algorithm. Furthermore, they have proposed a hybrid role mining approach that works with both the existing user-permission assignments and business information from the organization [92].

Molloy et al. have developed the *HierarchicalMiner*, an approach which considers both the semantics of roles and system complexity, and have shown that it is able to mine good roles as well as to generate good role hierarchies [89]. However, it does not consider the existing RBAC state and defines everything from scratch. Furthermore, it prunes the reduced concept lattice only based on wsc while the *StateMiner* does the pruning process based on GOF which is a global optimization function of weighted structural complexity and similarity. Also, the *HierarchicalMiner* does not consider the concepts with both new users

and new permissions in the pruning process, while the *StateMiner* considers them as well. There maybe a concept with both new users and new permissions that based on GOF, introduced in section 4.2.2.3, should be pruned. Moreover, if we set the weight factor for similarity to zero, $wf = 0$, our approach covers the *HierarchicalMiner* with the exception that it allows direct user permission assignment while our approach does not. In order to illustrate the strengths and weaknesses of nine different role mining algorithms, *Molloy et al.* introduce an evaluation framework for comparing them [90]. They also propose a new role mining algorithm and two new ways for algorithmically generating datasets for evaluating role mining algorithms.

Recently, *Vaidya et al.* have defined the *Minimal Perturbation problem* as "the problem of discovering an optimal set of roles from existing user permissions that are similar to the currently deployed roles" [88]. They also emphasize the idea of migrating to optimal RBAC as does our approach and use a similarity metric based on Jaccard coefficient to formalize the problem and propose a heuristic solution based on previously developed FastMiner algorithm [81]. However, their approach does not provide a complete solution to role migration as they just consider flat roles and ignore the role hierarchy and also according to their paper, the measure it uses to formulate similarity is very simple and not realistic. Furthermore, if in our approach we set the weights for wsc to $w_r = 1, w_u = w_p = 0$, and $w_h = \infty$, the measure will be only the number of roles. It means that we try to find the minimal set of roles, and that is the measure this approach uses. So, our approach can easily cover this approach as well.

Gue et al. have formally defined the notion of optimality for role hierarchy construction and also proposed a heuristic solution for that [86]. Their RH-Builder algorithm builds a hierarchy from the existing role set such that the number of direct relationships is minimized. They have also proposed the RH-Miner algorithm for the case where there are no initial roles. It separates the problem into two steps: first a minimal set of roles is generated using one of the known heuristic approaches, then the RH-Builder is applied to come up with the hierarchy with minimal number of edges. The key weakness of this approach is that it first generates a role set and then the hierarchy. Hence, the result may not really be optimal. Whereas, our approach integrates the role hierarchy creation and the discovery of roles.

4.0 THE PROPOSED SEMANTIC BASED POLICY MANAGEMENT FRAMEWORK

In this chapter, we present our proposed policy management framework. This framework aims to address the challenges discussed above and includes two main components: policy management and specification, and policy evolution. In the following sections, we present components of the proposed framework in detail.

4.1 POLICY MANAGEMENT AND SPECIFICATION COMPONENT

In this section, we present the policy management and policy specification component of the proposed framework. First, we present policy management as a service (PMaaS) framework and describe its components. Then, we introduce semantic based policy management framework which is an instance of the proposed PMaaS and aims to address heterogeneity issues. We introduce a semantic based policy specification in order to provide a common understanding basis for representations of concepts. This specification is able to support various policy specification models such as RBAC, hieratical RBAC, attribute based access control, group based policies and XACML, temporal constraints and restrictions of the GTRBAC model as well as collaboration among multiple CSPs. Finally, we present a proof of concept implementation of the proposed approach and present our performance evaluation results.

4.1.1 Policy Management as a Service (PMaaS)

In this section, we propose a cloud based policy management framework that puts users in control of their data that may be stored on multiple CSPs. It offers a centralized policy manager which enables users to specify their access control policies, no matter where the data is stored on the cloud.

We present the *policy management as a service* (PMaaS), a cloud-based framework that efficiently delivers policy management services. It is built on the concept of centrally expressing user's requirements that are applied to a user's data scattered across the cloud. Such requirements are expressed using access control policies to protect users' distributed resources. We define PMaaS as the capabilities provided to the customers to manage access policies for services and products running on a cloud infrastructure. The customers do not manage or control the underlying cloud infrastructure, network, servers, operating systems, storage, or even individual application capabilities. At a high level, the framework includes four main components: *cloud user*, *policy management service provider*, *cloud service provider (CSP)*, and *requester*. In the following, we provide a brief overview of each of these components.

- **Cloud User**: A cloud user uses different CSPs for various purposes. The cloud user is in charge of managing access policies on the policy management service provider which in turn will be used by CSPs to control access to the protected resources when a requester attempts to access them. The cloud user is also responsible for registering the CSPs at the policy management service provider so they can communicate the specified access policies.
- **Policy Management Service Provider (PMSP)**: A policy management service provider enables the cloud users to define, edit and manage their access policies. The cloud users can specify their policies using various models and specification languages which in turn are translated by the policy management service provider into a machine readable policy language. It is also responsible for conflict resolution on the policies to find and resolve possible conflicts and finally exporting the policies into target CSPs. Therefore, a policy management service provider acts as a Policy Administration Point

(PAP) and a Policy Information Point (PIP), as defined in [57].

- **Cloud Service Provider (CSP)**: A CSP offers one or more cloud services that are used by cloud users. A CSP controls access to the protected resources based on the policies specified by the cloud users. It evaluates access requests made by a requester against applicable policies and is in charge of making access decisions and enforcing those decisions when a requester attempts to access the protected resources. Therefore, a CSP acts as a Policy Decision Point (PDP) and Policy Enforcement Point (PEP), as defined in [57].
- **Requester**: A requester is an application controlled by a person or a company that interacts with a CSP in order to get access to a protected resource belonging to the specific cloud user. It can be a CSP that accesses resources stored in another CSP.

Figure 2 illustrates the framework and its components in detail which are described in the following sections.

4.1.1.1 The Cloud Service Provider (CSP) As shown in Figure 2, each CSP keeps a repository of all the resources that cloud users store and has its own access control system that makes decisions and enforces them based on input from the policy management service provider. PMaaS does not impose any restrictions on what access control model the CSPs use. It means that each service provider has its own policy engine and may use a simple access control matrix or a complex flexible policy engine. Each CSP has also a local policy base to store policies and an authorization API that is used by the policy management service provider to export access policies into the CSPs.

4.1.1.2 The Policy Management Service Provider (PMSP) The policy management service provider (PMSP) is the most important part of the framework and as shown in Figure 2, has two main components, the policy editor and the policy server.

The policy editor acts as Policy Administration Point (PAP) and provides interfaces for cloud users to manage access policies in a single centralized location. It facilitates the policy management process for cloud users by allowing them to specify their policies in natural language. It also handles the CSP registration process which will be explained later.

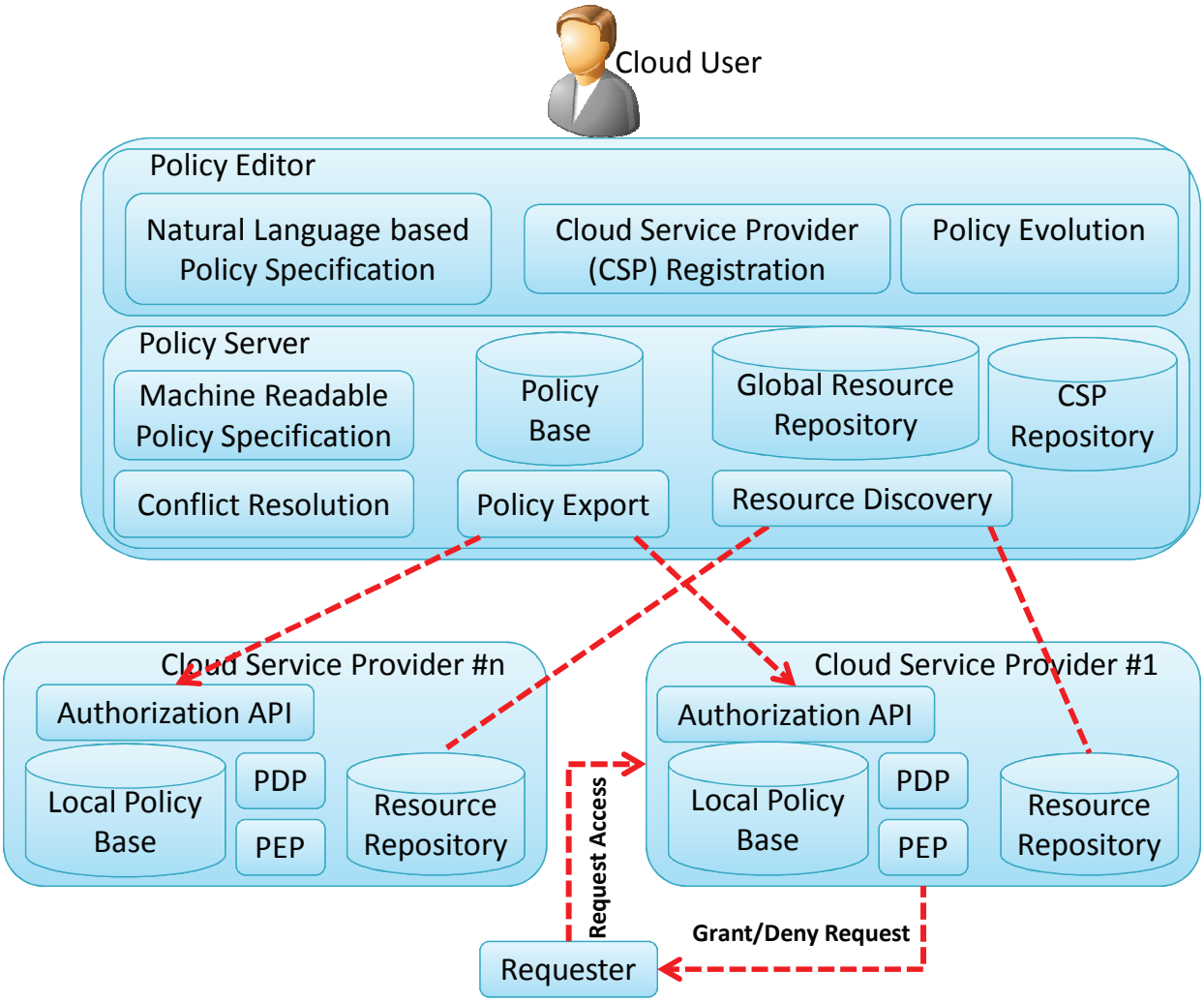


Figure 2: The Proposed Policy Management as a Service (PMaaS) Framework

As policies change and evolve over time, we need an approach to deal with it. The policy evolution unit is responsible for addressing this issue and we use role mining techniques for this purpose. It will be discussed in detail in section 4.2.

The policy server acts as Policy Information Point (PIP) and is responsible for interacting with the policy editor and the CSPs as well as translating the policies specified by the cloud user into a machine readable policy language. It keeps a repository of registered CSPs associated with each cloud user. It is also responsible for the resource discovery process which we will explain in Section III.C. After the cloud user registers its CSPs at the PMSP, the PMSP communicates with each CSP to find resources and stores them in a global resource repository which contains all resources and their association with cloud users and CSPs. These resources are presented in policy editor interface to the cloud user to help him/her in specifying policies.

Moreover, it receives the policies specified by cloud user in policy editor, parses them, transfers them into machine readable policy language and stores them in a policy base. The output policy language could be XACML [62] or an OWL-based policy language such as Rei (http://rei.umbc.edu). Since there is no one agreed-upon policy language that all CSPs use, the framework should have capability to provide the output policies in multiple languages to support as many CSPs as possible.

Next, the policy server detects and resolves possible conflicts among access policies. And the final step is to export policies into the CSPs; the policy server first separates the policies related to each CSP based on the resource-provider association and export them into the associated CSP using its authorization API.

4.1.1.3 The Communication Protocols At a high level, interactions among components of the proposed framework include the following steps: *registering CSPs*, *resource discovery*, *specifying policies*, and *exporting policies*. In the following sections, we discuss each of these steps in more detail.

Step 1: Registration of CSPs at PMSP

In this step a cloud user registers all CSPs he/she uses at the policy management service provider. As shown in Figure 3, this can be achieved by providing the location of the CSP

to PMSP.

When the location of the CSP is provisioned to the PMSP, it uses the host-meta discovery mechanism, such as one proposed in [65], to obtain a host-meta document from the CSP. Such a document defines the location of a user authorization URL among other items. The PMSP then uses the user authorization URL to initiate the process of acquiring authorization to communicate with particular CSP. When a PMSP receives the host-meta document from the CSP, it obtains the cloud user's authorization to communicate with this CSP. This is achieved by receiving a verification code authorized by resource owner from the CSP. This verification code is generated for purpose of authorizing the PMSP to be able to communicate with the CSP. At the end of this step, a PMSP is able to communicate with the CSP to discover resources and also export access policies specified by the cloud user.

Step 2: Resource Discovery by PMSP

After a cloud user registers CSPs at the PMSP, we have a repository of all CSPs at the PMSP. Next, the PMSP communicates with each of the CSPs to discover the resources stored in them.

Resources are identified using Uniform Resource Identifiers (URIs) which are compact sequences of characters that identify an abstract or physical resource and provide a simple and extensible means for identifying a resource [66]. The URI syntax only allows a subset of ASCII, about 60 characters. Internationalized Resource Identifiers (IRIs) are a new protocol element, a complement to URIs that were developed to address limitations of the URI system [67]. An IRI is a sequence of characters from the Universal Character Set (Unicode/ISO10646). There is a mapping from IRIs to URIs, which means that IRIs can be used instead of URIs where appropriate to identify resources [67].

Similar to IRIs, Extensible Resource Identifiers (XRIs) extend the syntactic elements allowed in IRIs [68]. XRIs provide a standard syntax and resolution protocol for abstract identifiers compatible with URIs and IRIs. The goal of XRI is to provide a standard syntax and discovery format for abstract, structured identifiers that are domain-, location-, application-, and transport-independent, so they can be shared across any number of domains, applications, directories, and interaction protocols. It is built directly on top of the foundation provided by the URI and IRI and its syntax specification is based on the URI

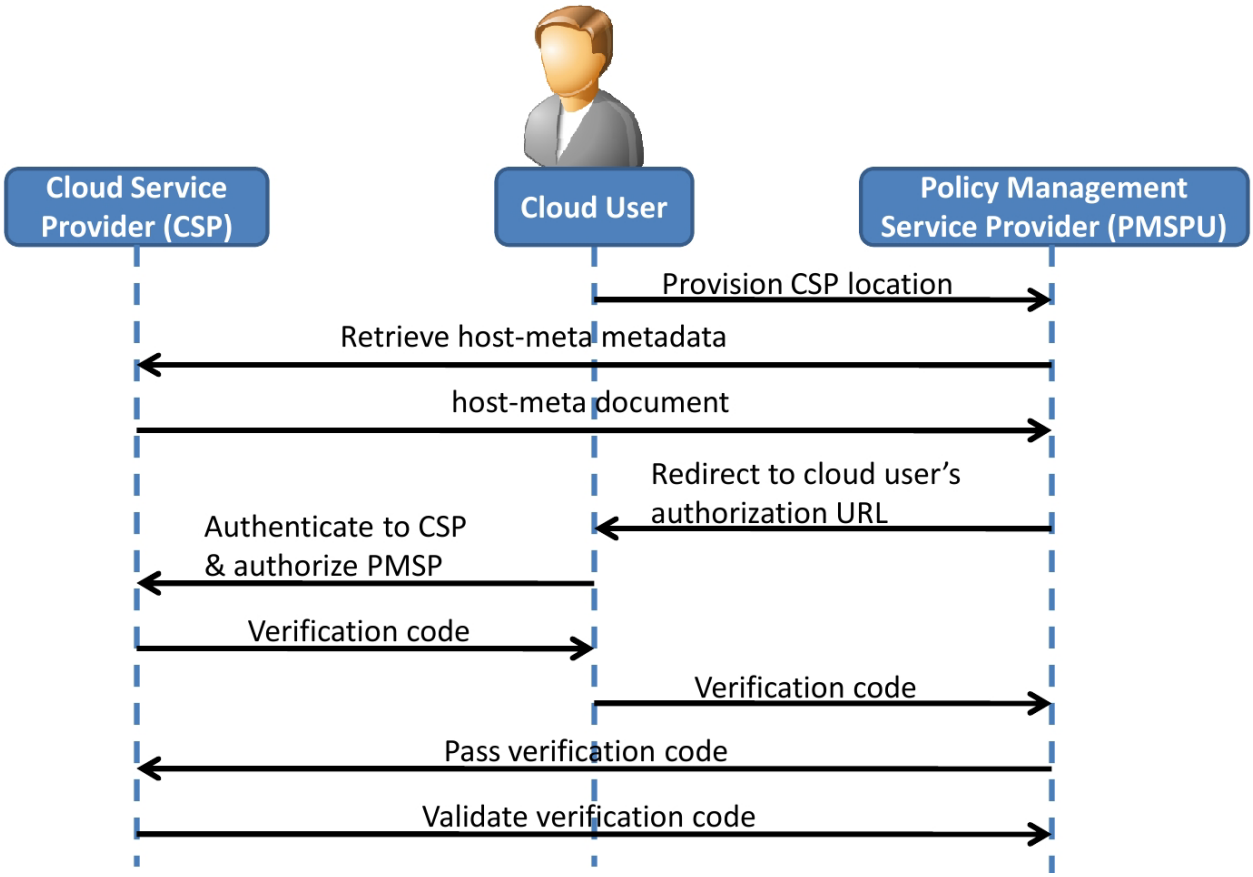


Figure 3: CSP Registration Process

specification and the IRI specification. To support applications that expect IRIs or URIs, the XRI specification also defines rules for transforming an XRI reference into a valid IRI or URI reference. It also includes a simple XML descriptor format and HTTP(S) protocol for uniform resource metadata discovery. Extensible Resource Descriptor (XRD) is a simple generic format for describing resources [69]. It provides machine-readable information about resources for the purpose of promoting interoperability.

However, we recommend using of POWDER-S or Semantic POWDER [64] for resource description. The Protocol for Web Description Resources (POWDER) "facilitates the publication of descriptions of multiple resources such as all those available from a Web site" [63]. POWDER documents are written in XML and have loose semantics, however, POWDER-S has been developed to support Semantic POWDER of Description Resources.

For the CSPs that do not support/use POWDER-S, host-meta discovery mechanism [65] can be used for resource discovery. It is a lightweight meta-data document format to be used for describing hosts and intended for use by web-based protocols [65]. It contains information about individual resources controlled by the CSP.

Step 3: Specification of Access Policies at PMSP

Next step is to specify access policies that are applied to resources. Our framework does not impose any constraints on how cloud users specify access policies. There are various specification languages and models that could be used for this purpose. In section 4.1.3, we will introduce how OWL based specification can be used for specifying access control policies in cloud computing environments.

Alternatively, it could allow cloud users to specify their policies in natural language. There has been some efforts to allow users to specify policies in controlled natural language [61]. IBM's SPARCLE tool aims to enable users to enter policy rules in natural language. It automatically parses the policies to identify policy elements. Then, it generates a machine readable version of the policies that can be used by any enforcement engine that can handle the standardized XML format. The policy creation portion of SPARCLE provides visualization features to help the users ensure that the specified policies are what they intended them to be.

The same system can be adopted to be used in our framework for policy specification.

However, it uses structured entry methods and guided natural language which is a kind of controlled natural language for policy specification. Controlled natural languages (CNLs) are "subsets of natural languages, obtained by restricting the grammar and vocabulary in order to reduce or eliminate ambiguity and complexity" [71].

Step 4: Translation of access policies by PMSP and exporting them into CSPs

After the access policies are specified in controlled natural language or other supported formats, the PMSP parses the policy, identifies policy elements, and transforms the policy into machine readable language. Next, a conflict detection and resolution is done on the policies to remove potential conflicts. Then, using the association between resources and CSPs, the policies are separated based on target CSP. Finally, the access policies are exported into their related CSP using an authorization API.

In order to export policies into the CSPs, we propose to use the W3C Rule Interchange Format (RIF) which is a format to exchange rules between rule engines that operates over both XML and RDF data [70]. It is a standard format for exchanging rules among rule systems, particularly in Web rule engines. The central idea behind the exchange of rules through RIF is that different systems provide syntactic mappings between their native languages and RIF dialects [70]. The systems can communicate through an appropriate dialect, which they both support. In order to be able to communicate rule sets from one system to another, the mappings should be semantics-preserving. Due to its extension mechanisms, RIF is an ideal language to investigate machine-readable first-order logic rules.

In general, for the PMSP to be able to export policies into the CSPs, it should be able to exchange data with the CSPs. The ideal situation for PMSP is that the CSPs support well known policy languages such as XACML and provides required APIs to exchange data with the PMSP. However, some CSPs may not be willing to do changes and may not support well known policy languages. In this case, it is the responsibility of the PMSP to generate policies in a format that the CSP supports. It is clear that the more machine readable formats the PMSP provides and the more CSPs offer APIs, the easier PMaaS would be deployed.

The requester sends access requests to the CSP, and the CSP handles them locally. Our proposed framework does not impose any limitations on the requester and the CSP's decision making functionality. The requester can send a request to access protected resources similar

to any system. The CSP checks the request against its policy base; makes decision, and grants or denies the access to the requester based on access policies defined by the cloud user.

Whenever a cloud user changes his policies, the PMSP applies the required updates and communicate them with the target CSPs. This could be done by *push* or *pull* strategy. In *push* strategy, whenever there is a change, the PMSP updates the policies and exports them to target CSPs while in *pull* strategy, the CSP initiates the communication and checks for possible policy changes/updates in certain time periods. However, we believe that *push* strategy is more efficient in this situation. Since the associations between policies, resources, and CSPs have been already identified, the PMSP only needs to relate the changes to target CSPs and export the updated policies into them.

Similarly, whenever cloud users add/remove resources to/from CSPs, appropriate updates need to be done. However, in this case we believe that *pull* strategy is better because resources are stored in CSPs and whenever there is a change, CSPs can inform the PMSP to get updated policies.

The question may arise regarding privacy of cloud user's identity and privacy of his policies and the concern is that cloud users have to trust PMaaS provider to provide their identifications in different CSPs and specify their access policies using the PMaaS. We assume that there is enough level of trust between cloud user and PMaaS provider to deploy the service. However, the PMaaS could be deployed as private cloud within an organization's premise or fully controlled by an individual user to avoid privacy concerns.

4.1.1.4 Discussion

In proposing this framework, we have tried to keep required changes at the CSP side as minimal as possible. This way, we can make sure that more CSPs would be willing to use the proposed framework and very little effort will be required to deploy and use it. The only thing the CSP needs to be able to use the proposed framework is to provide an API for exchanging data with the PMSP and information about the policy format it supports so that the PMSP knows what format the access policies should be translated to.

Our proposed cloud based policy management framework has some advantages over

existing systems as discussed in the followings.

- Access policy specification functionality is externalized from CSPs and can be done in a centralized location for all cloud providers. Decisions about who has access to what resources are made locally and enforced by each CSP. However, the specification of policies for all resources and services is done centrally in a single location.
- Cloud users use a unified policy management system to control access to all their resources scattered over the Cloud. They do not need to deal with various policy management systems bound to each CSP.
- Cloud users use a single management tool to compose access policies which allows them to have a consistent user experience when managing these policies. They do not need to learn to work with different interfaces and tools.
- Since access policies are composed using a single policy management tool and hosted in a single location, cloud users have a consolidated view of the access policies applied to their resources. It is easier for users to introduce new access policies and modify existing policies when needed.
- If cloud users move their resources from one CSP to another for any reason, they do not need to redefine all the policies again. For example, if Alice moves one document from *GoogleDocs* to *Amazon S3*, she does not need to redefine the policies associated with that document in *Amazon S3*.
- With existing systems, the cloud user is limited to the functionality provided by the CSPs' policy engine. However, our proposed framework may be able to apply extra policies by transforming them into the provider's policies. For instance, in *Facebook* Alice can share her location but she is not able to define any temporal constraint on that. If she wants to share her location at some specific times, she can not do it in *Facebook* but she can do it using our proposed framework. She can specify a policy that her location should be private between 8 am and 5 pm everyday otherwise it can be shared with friends. Our proposed framework can export a policy into the *Facebook* at 8 am that makes Alice's location private and when the time is 5 pm, it can export another policy into the *Facebook* to share her location with friends.

The proposed framework is extensible and flexible; it is designed in a way that it can support various specification languages as long as it can translate them to a set of machine readable languages supported by the framework and understandable by the CSPs. It can also be extended to include policy decision point too and perform access decisions and send them to the CSPs for enforcement. We can also extend the PMaaS by offering policy recommendation capabilities where the framework can analyze the policies defined by user and use machine learning techniques for example to predict policies and recommend them to user. Of course, user will have the option to reject or modify the recommended policies.

4.1.2 Semantic Based Unified Policy Management Framework

In this section, we introduce a semantic based policy management framework which is an instance of the previously proposed PMaaS framework. The framework also addresses interoperability and heterogeneity issues as well as other requirements we discussed.

In order to deal with the heterogeneity of cloud computing environments, representations of concepts and access policy rules should be flexible and generic to fulfil various cloud providers' modeling requirements. We use Semantic Web technologies to address these representations and model concepts and semantics of different cloud providers with high expressiveness. The proposed framework should provide capabilities to users to manage access policies for services running on a cloud infrastructure.

In our proposed approach, we use the combination of OWL 2 RL and SWRL with DL-Safe restriction to express and manage authorization policies; they are referred as OWL and SWRL respectively in the rest of this dissertation. This combination offers the following advantages:

- It offers high expressiveness by providing a wide variety of constructors to represent knowledge.
- Its reasoning could be implemented using rule-based engines with good performance.
- It provides scalable reasoning capability without having to sacrifice too much expressive power.

- It helps to address heterogeneity management and interoperability issues among different CSPs.
- It provides separation between policy description and domain description.

In the following, we present the proposed *semantic based policy management framework* for cloud computing environments that delivers policy management services.

Our proposed semantic based policy management framework is built on the concept of centrally expressing a user's security requirements that are applied to a user's resources scattered across the cloud. The customers do not manage or control the underlying cloud infrastructure, network, servers, operating systems, storage, or even individual application capabilities. Figure 4 illustrates the framework and its components which are briefly described in the followings.

At a high level, the two components of the proposed architecture are the *cloud service provider (CSP)* and the *semantic based policy management service*.

- A cloud service provider (CSP) offers one or more cloud services that are used by cloud customers and controls access to the protected resources. It evaluates access requests made by a requester against applicable policies and is in charge of enforcing access decisions when a requester attempts to access the protected resources. Therefore, a CSP acts as a policy decision point (PDP) and policy enforcement point (PEP) [57].
- A semantic based policy management service (SBPMS) enables the cloud users to specify, edit and manage their access policies. It is also responsible for conflict resolution on the policies and exporting the policies into target CSPs. Therefore, SBPMS acts as policy administration point (PAP) and a policy information point (PIP) [57].

Each CSP includes a *semantic based policy decision point (PDP)*, a *policy enforcement point (PEP)* and its own *local knowledge base*. The local knowledge base stores all the ontologies and the policy rules of the CSP. The semantic based PDP component is in charge of making authorization decisions while the PEP module is in charge of enforcing those decisions.

The SBPMS provides authorization services and interfaces for cloud users to manage access policies in a single centralized location. Once the policies are specified by cloud users, it detects and resolves possible conflicts among access policies. Then, it exports policies

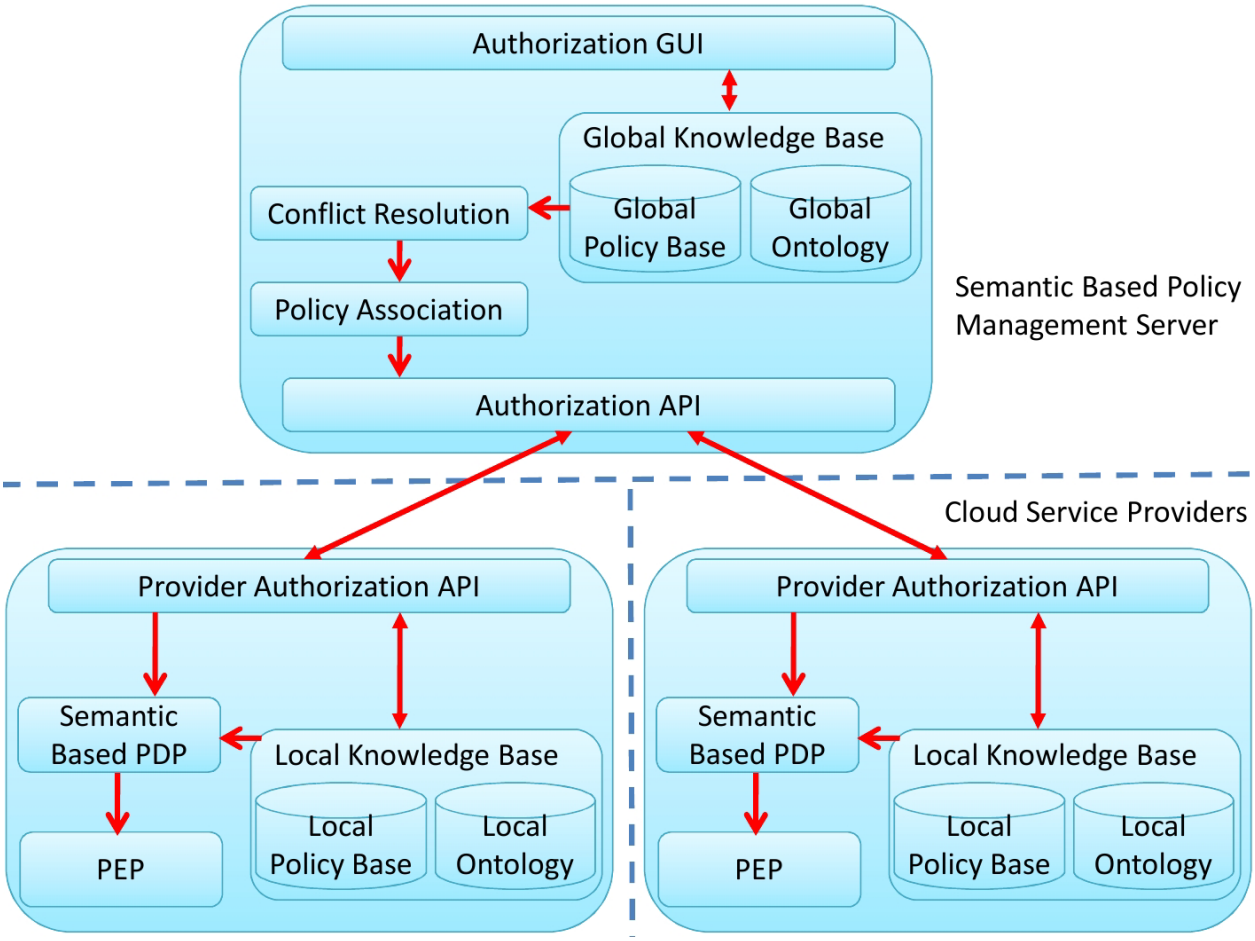


Figure 4: The Proposed Semantic Based Policy Management Framework

into the CSPs. In order to do this, it first separates the policies related to each CSP based on the resource-provider association and then exports them into the associated CSP via the *provider authorization API*. The SBPMS has several components. The *global knowledge base* is a central repository that stores all the ontologies and the policy rules gathered from different CSPs' local knowledge bases. This is done using the *provider authorization API*s.

The *authorization GUI* provides users with information that is required for specifying their access policies that are retrieved from the global knowledge base. The *conflict resolution* module then detects and resolves possible conflicts among the specified access policies while the *policy association* module is responsible for associating the policies with their target CSPs and services. An *authorization API* provides services related to authorization management process and it is accessed via the provider authorization API.

The most important component of the framework is the SBPMS. It handles the multiple ontologies and access policies of the CSPs and uses that knowledge in policy specification. Each CSP has its own knowledge base that includes description of its users, services, resources and any other information related to the domain. The SBPMS requires CSPs to provide such information for authorization purposes. In order to enable the SBPMS to perform the semantic based authorization process, the CSPs represent information in OWL 2 ontologies and SWRL rules and keep it up to date.

Whenever a cloud user changes his policies, the SBPMS applies the required updates and communicates them to the target CSPs. This could be done by *push* and/or *pull* strategies. In the *push* strategy, whenever there is a change, the SBPMS updates the policies and exports them to target CSPs while in the *pull* strategy, the CSP initiates the communication and checks for possible policy changes/updates in certain time periods. We believe that *push* strategy is more efficient in this situation. Since the associations among policies, resources, and CSPs have been already identified, the SBPMS only needs to relate the changes to target CSPs and export them.

Similarly, whenever cloud users add/remove resources to/from CSPs, appropriate updates need to be done. In this case, we believe that *pull* strategy is better because resources are stored in the CSPs and whenever there is a change, CSPs can inform the SBPMS to get updated policies.

A question may arise regarding the privacy of cloud user's identity and privacy of his policies. The concern here is that cloud users have to trust SBPMS to provide their identifications in different CSPs and specify their access policies using the SBPMS. We assume that there is enough level of trust between cloud user and the SBPMS to deploy the service. In order to avoid privacy concerns, however, we can use cryptographic methods. We can use encrypted ontologies, encrypted ontology-mapping and encrypted queries as explained in [37]. Alternatively, the SBPMS could be deployed as private cloud within an organization's premise or be fully controlled by an individual user.

The key advantage of our proposed framework is that with existing systems, the cloud user is limited to the functionality provided by the CSPs' policy engine. However, our proposed framework may be able to apply additional policies by transforming them into the provider's policies. For instance, in *Amazon S3* or *Dropbox*, if Alice wants to share a file with a group of users, she has to specify different policies for each user. In our system, however, she can specify a policy to share the file with a group of users called colleague and our system exports required policies (one policy per each user of the group) into the *Amazon S3* or *Dropbox*.

The OWL and SWRL reasoner in the semantic based PDP component performs a reasoning process to take access decisions [34]. The reasoner includes three main operations: *inference*, *validation*, and *querying the ontology* as described below.

- Inference: Using the information available in the ontology helps infer new knowledge about the CSP. The results of inference on SWRL rules are OWL instances which are used in the access decision making process.
- Validation: The validation operation can detect whether constraints expressed using OWL 2 language are violated by looking for possible inconsistencies.
- Querying the ontology: This operation is used for inheritance recognition and instance recognition. The instance recognition tests whether an individual in the knowledge base is instance of a class expression whereas the inheritance recognition examines if a property is sub-property of another property or if a class in the knowledge base is subclass of another class. Using this operation, we can formulate generic queries that refer to abstract concepts and consequently the system can recognize instances that belong to subclasses

or sub-properties of such an abstract concept.

One of the important functionalities of a reasoner is the ability to detect and resolve conflicts [34]. There are two main categories of conflicts in policy evaluation: semantic conflicts and syntactic conflicts. The semantic conflicts are results of using information that is relevant to the current state of the system. So their appearance depends on the dynamic state of the domain and they are really difficult to detect. The syntactic conflicts, however, are caused by specification errors in the policy or derived from other rules. Their appearance does not depend on state of the application domain and we can detect them by examining structure of the rules.

Furthermore, the heterogeneous environments of cloud computing composed of different CSPs may lead to more policy conflicts among different CSPs' policies. One of the syntactic conflicts is modality conflicts that occur if two or more policies refer to the same subjects, objects and actions and have modalities of opposite sign. This type of conflicts can be detected from the syntax of the rules that specifies access policies.

In our proposed system, if two disjoint properties appear at the same time, the reasoner identifies a policy conflict. In order to do this, we define an *unauthorizedSubject* association using the *disjointDataProperties* construct provided by OWL and declare it as disjoint with *authorizedSubject*. For example, a modality conflict happens if two different policies permit and forbid *Subject1* to do *Action1* on *Object1*. If the reasoner infers an instance *unauthorizedSubject* between *Subject1* and *Object1*, due to some policy and at the same time, it also infers an instance of *authorizedSubject* between them, a conflict occurs.

Semantic conflicts, on the other hand, can be detected using some meta-policies that describe unacceptable situations in the domain. One situation where this kind of conflicts may occur is when a subject can change its own permissions. For instance, if *Subject1* is allowed to change his own permissions, he may be able to perform *Action2* on *Object1* which initially was forbidden for him. We propose to adopt approaches based on meta-policy to detect such a situation [39].

Once the system detects conflicts, a resolution strategy is needed to provide a solution for resolving detected conflicts. We can use policy prioritization as a solution to resolve the authorization conflicts. If the conflicts occur between policies specified by different CSPs, we

assign a priority to each CSP to resolve the conflict. If the conflicts are between policies of the same CSP, we assign priorities to the rules or the authorization decisions. For example, we can state that negative policies have priority over positive policies.

We introduced how our proposed framework can detect and resolve both semantic and syntactic conflicts. However, detecting and resolving conflicts are complex tasks which are beyond the scope of this dissertation.

4.1.3 Semantic Based Policy Specification

Using Semantic Web languages like OWL 2 and SWRL provides us with powerful expressiveness to specify access policies and to satisfy the modeling requirements of various CSPs. However, for the policy management service to be able to understand the policies, we need to define a set of concepts [34]. These concepts are specified in a way that the policy management service can provide an authorization statement, positive or negative. Note that the definition of such concepts does not restrict the CSPs' ontologies. The CSPs can either use this defined set of concepts or define their own concepts. In the latter case, however, they should provide some OWL constructors and/ or SWRL rules to map their concepts to the ones required by the policy management service to perform the authorization process.

4.1.3.1 General Semantic Based Policy Specification In this section, we present semantic based policy specification that provides a common understanding basis for policy specification in cloud computing environments. As already mentioned we use OWL to model this specification. We use an ontology to model and unambiguously represent all the entities involved in an authorization decision process such as subject, object, action and their attributes within cloud computing environments.

Such an ontology provides a basis for addressing heterogeneity management issues arising from data owners dealing with several CSPs and facilitate collaboration among various CSPs. It provides capabilities for data owners to specify access policies and is tailored towards cloud computing environments where data owners may have resources on multiple CSPs.

The ontology is designed to be used with the semantic based policy management frame-

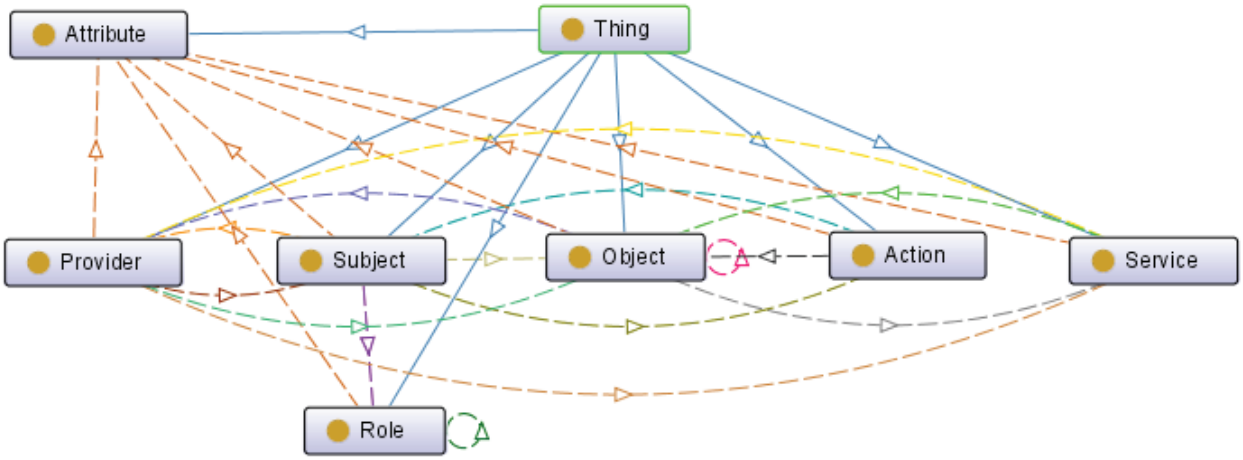


Figure 5: The Proposed Ontology

work and tries to enable data owners to specify policies that are applied to their resources scattered across various CSPs which might have different access control mechanisms. It enables users to define policies that are understandable by various CSPs which may be using different access control mechanisms.

The ontology is flexible and able to model various access control models such as RBAC, hierarchical RBAC, attribute based access control, group based access control, hierarchical object. Considering the use case scenarios described in section 2, data owners can use this ontology to define access policies for their resources which are stored on several CSPs.

Ontology We define an ontology that are used in policy specification and management as shown in Figure 5. The subjects, objects, actions and other entities in the system are modeled as an OWL class as follows:

Subject rdfs:subClassOf owl:Thing

Role rdfs:subClassOf owl:Thing

Object rdfs:subClassOf owl:Thing

Action rdfs:subClassOf owl:Thing

Attribute rdfs:subClassOf owl:Thing

Provider rdfs:subClassOf owl:Thing

Service rdfs:subClassOf owl:Thing

Subject Ontology

A subject is an active entity that has permissions to perform some actions over objects. In the cloud computing environments, a subject could be a user, a user group, a role, a process, a service and so on. The subjects are modeled as an OWL class Subject. The instances of this class represent the subjects on which the policies are defined. The object property and data property of OWL are used to describe attributes of a subject with *hasSubjectAttribute* and *hasSubjectDataAttribute* respectively.

In order to support RBAC, we add the concept of the role to the subject ontology. A subject's capability to perform a task is represented by the role in subject ontology. The role represents access permissions of resources and similar to RBAC model, a subject can access the resources by being assigned to a role. We can use ObjectProperty *hasRole(Subject, Role)* to assign a role to a subject. Alternatively, we can describe subject attributes and attribute requirements of role and if a subject satisfies the requirements of the role attributes, it can be assigned to that role. The latter approach is more efficient and is able to provide dynamic role assignment. We describe a subject performs an action on an object using tow properties *performsAction(Subject, Action)* and *accessObject(Subject, Object)*. Moreover, we can link a subject to a provider by the ObjectProperty *isAssociatedWithProvider(Subject, Provider)*.

Subject rdfs:subClassOf owl:Thing

hasRole a rdfs:Property, owl:ObjectProperty

rdfs:domain :Subject;

rdfs:range :Role.

performsAction a rdfs:Property, owl:ObjectProperty

rdfs:domain :Subject;

rdfs:range :Action.

accessObject a rdfs:Property, owl:ObjectProperty

rdfs:domain :Subject;

rdfs:range :Object.

isAssociatedWithProvider a rdfs:Property

, owl:ObjectProperty

rdfs:domain :Subject;

rdfs:range :Provider.

In order to support hierarchical RBAC, we introduce a hierarchical subject grouping into the policy management framework. It allows to define hierarchical roles where a role can be defined as a specialization of another role. These rules can extend the scope of an action to a given subject to all the roles that are descendants of the original role assigned to the subject. The role hierarchy is captured using OWL property *subRoleOf*. Furthermore, we define *subRoleOf* to be transitive by making it an instance of *owl:TransitiveProperty* class. In this way, a reasoner can infer that if $subRoleOf(role_i; role_j)$ and $subRoleOf(role_j; role_k)$, then $subRoleOf(role_i; role_k)$.

Role rdfs:subClassOf owl:Thing

subRoleOf a rdfs:Property

, owl:TransitiveProperty

rdfs:domain :Role;

rdfs:range :Role.

RoleMappingEnabled a rdfs:Property

rdfs:domain :Role;

rdfs:range :Role.

conflict a rdfs:Property

rdfs:domain :Role;

rdfs:range :Role.

prerequisite a rdfs:Property

rdfs:domain :Role;

rdfs:range :Role.

isAssociatedWithProvider a rdfs:Property

, owl:ObjectProperty

rdfs:domain :Role;

rdfs:range :Provider.

Object Ontology

Object is an entity that is accessed and/or modified by a subject. In the cloud computing environments, an object could be documents, data, services or other resources. The object property and data property of OWL are used to describe attributes of an object with *hasObjectAttribute* and *hasObjectDataAttribute* respectively. There is no actual difference between an object and a subject. A subject entity in one CSP might be object entity in another CSP and vice versa.

Similar to the concept of role in the subject ontology, object group is defined to organize the objects. Each object group is a new concept in the ontology which includes multiple individual objects. All individual objects associated with the object group concept will have the same object attribute values as the object group. The object group can be formally defined as follows:

$ObjectGroup_i \equiv \{Object_1, ..., Object_n\}$ where $\forall j (j = 1, 2, ..., m)$:

$Object_1.hasObjAtt_j.AttributeValue =$

$... =$

$Object_n.hasObjAtt_j.AttributeValue =$

$ObjectGroup_i.hasObjAtt_j.AttributeValue$

where $hasObjAtt_j$ is the sub property of object property *hasObjectAttribute* or data property *hasObjectDataAttribute*.

Object hierarchies help in the policy management by extending the scope of an action on an object to all the descendant objects of the original object. The object hierarchy is captured using OWL property *isDescendantOf*. Furthermore, we define *isDescendantOf* to be transitive by making it an instance of *owl:TransitiveProperty* class. In this way, a reasoner can infer that if $isDescendantOf(Obj_i; Obj_j)$ and $isDescendantOf(Obj_j; Obj_k)$, then $isDescendantOf(Obj_i; Obj_k)$.

Object rdfs:subClassOf owl:Thing

isAssociatedWithService a rdfs:Property

, owl:ObjectProperty

rdfs:domain :Object;

rdfs:range :Service.

isOwnedByProvider a rdfs:Property

, owl: ObjectProperty

rdfs:domain :Object;

rdfs:range :Provider.

isDescendantOf a rdfs:Property

, owl:TransitiveProperty

rdfs:domain :Object;

rdfs:range :Object.

Action Ontology

Actions are defined based on the type of the actions that subjects can perform on objects. Each action type is a concept in the ontology and the actions are individuals of the concept. Although number of subjects and objects is large, number of actions is usually very small. Examples of actions are read, write, execute and so on. As previously mentioned, CSPs collaborate and interoperate, therefore a lot of procedures may be processed in parallel. For example, one CSP may perform read action on a resource while another CSP is performing write action on the mentioned resource at the same time. We add some actions such as parallel write, parallel read, parallel execution, parallel read and write, and so on to capture these parallel procedures in the cloud computing environments. The *hasActionAttribute* property is used to describe relevant information of action to be used for authorization management purposes.

We also define action group similar to object group that could be useful in the definition of rules. The action group can be formally defined as follows:

$ActionGroup_i \equiv \{Action_1, ..., Action_n\}$ where $\forall j (j = 1, 2, ..., m)$:

$Action_1.hasActAtt_j.AttributeValue =$

$... =$

$Action_n.hasActAtt_j.AttributeValue =$

$ActionGroup_i.hasActAtt_j.AttributeValue$

where $hasActAtt_i$ is the sub property of object property *hasActionAttribute*.

The Action class binds a subject to an object using two properties *hasSubject(Action, Subject)* and *hasObject(Action, Object)*.

Action rdfs:subClassOf owl:Thing

hasSubject a rdfs:Property, owl:ObjectProperty

rdfs:domain :Action;

rdfs:range :Subject.

hasObject a rdfs:Property, owl:ObjectProperty

rdfs:domain :Action;

rdfs:range :Object.

Provider Ontology

Provider is the entity that offers one or more services to customers and controls all the objects related to those services. It essentially controls what actions each subject is allowed to perform on its objects. We use the object property and data property of OWL to describe attributes of an provider with *hasProviderAttribute* and *hasProviderDataAttribute* respectively.

Provider rdfs:subClassOf owl:Thing

providesService a rdfs:Property, owl:ObjectProperty

rdfs:domain :Provider;

rdfs:range :Service.

ownsObject a rdfs:Property, owl:ObjectProperty

rdfs:domain :Provider;

rdfs:range :Object.

isAssociatedWithSubject a rdfs:Property

, owl:ObjectProperty

rdfs:domain :Provider;

rdfs:range :Subject.

Service Ontology

Service is the entity that is offered by a provider and includes some objects that are ac-

cessed/modified by subjects. The service is associated with objects. The object property and data property of OWL are used to describe attributes of an object with *hasServiceAttribute* and *hasServiceDataAttribute* respectively.

Similar to object, we define service groups to organize them. Each service group is a new concept in the ontology which includes multiple individual services. All individual services of the service concept have attribute values of the service group. The service group can be formally defined as follows:

$ServiceGroup_i \equiv \{Service_1, ..., Service_n\}$ where $\forall j (j = 1, 2, ..., m)$:

$Service_1.hasSrvAtt_j.AttributeValue =$

$... =$

$Service_n.hasSrvAtt_j.AttributeValue =$

$ServiceGroup_i.hasSrvAtt_j.AttributeValue$

where $hasSrvAtt_i$ is the sub property of object property *hasServiceAttribute* or data property *hasServiceDataAttribute*.

Service rdfs:subClassOf owl:Thing

isAssociatedWithObject a rdfs:Property

, owl:ObjectProperty

rdfs:domain :Service;

rdfs:range :Object.

offeredBy a rdfs:Property

, owl:ObjectProperty

rdfs:domain :Service;

rdfs:range :Provider.

Attribute Ontology

The attribute ontology defines attribute types that can be used to define the attribute of entities such as subject, object and action. We need the attribute value of entities in authorization process to decide whether entities meet the authorization conditions to permit or deny request. There is a partial order among attribute values in attribute definition which is defined based on the numerical values of the attribute values of data property. However,

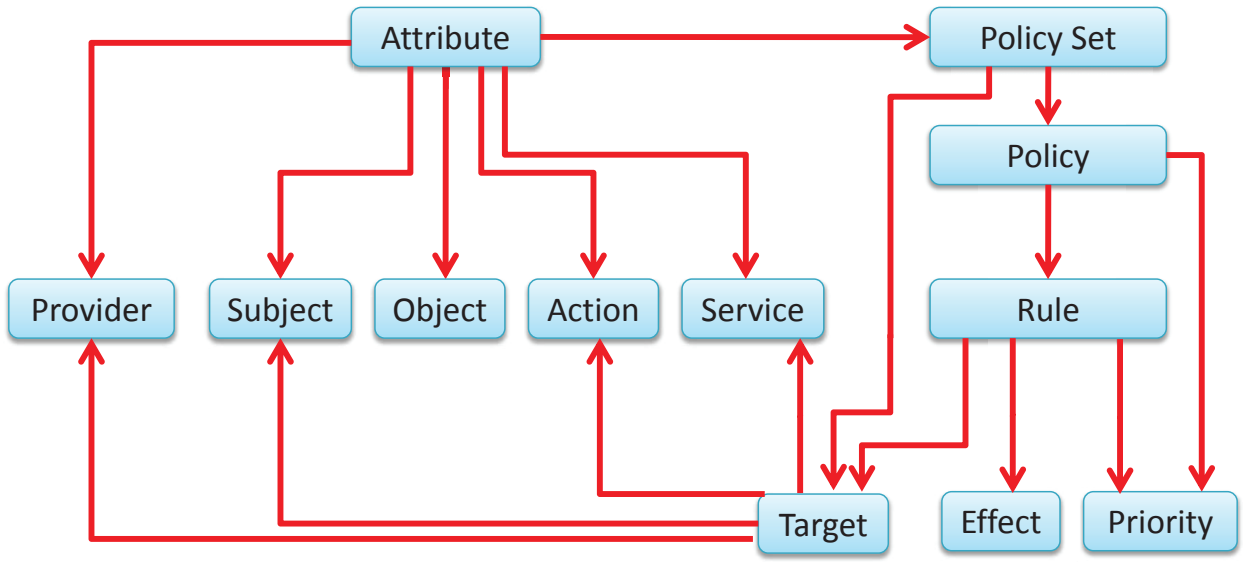


Figure 6: The Policy Specification Meta Model

for the non-data properties of attribute values, we need to manually define partial order among attribute values.

Policy Rules

A typical policy rule is presented by a 3-tuple *[Subject, Object, Action]*. We extend this 3-tuple to support interoperation in the cloud computing environments. We define a 5-tuple *[Provider, Subject, Object, Action, Service]* where provider represents the CSP which is associated with the subject and service represents cloud service which is associated with the object. This 5-tuple is interpreted as follows: the *Subject* associated with the *Provider* is allowed to perform the *Action* over the *Object* associated with the *Service*.

The provider part of the rule could be any CSP that stores objects and offers some services. The subject of the rule could be a single user, a user group, or role of any CSP. The object part of the rule can be a single object or object group controlled by CSPs. The service part of the rule can be a service or service group provided by any CSP.

As shown in Figure 6, a typical access rule in our system includes zero or one target, an effect, and zero or one priority. The target of the rule is presented by a 5-tuple *[Provider, Subject, Object, Action, Service]* as described above. The effect part of a rule is the autho-

rization result allow or deny.

Policy is to determine how to compose different rules and policy set is to determine how to compose different policies. There are different approaches that can be used for this purpose such as deny override, permit override and first applicable algorithms recommended by XACML [62].

As mentioned before, by having role ontology we can represent access control policies of flat RBAC and hierarchical RBAC. We can also extend the proposed ontology to represent various constraints in RBAC. We define a new entity, *session* to represent the sessions in RBAC; users can *establish* sessions and roles that are activated in a session can be modeled using *hasActiveRole* property. We should define rules for various types of constraints. For example, the following rules represent session constraint rule, prerequisite role constraint rule and mutually exclusive role constraint rule [45].

- $establish(?u, ?s) \wedge subRole(?u, ?r) \rightarrow hasActiveRole(?s, ?r)$
- $hasRole(?u, ?r) \wedge prerequisite(?r^{'}, ?r) \rightarrow \neg hasRole(?u, ??r^{'})$
- $hasRole(?u, ?r) \wedge conflict(?r, ?r^{'}) \rightarrow \neg hasRole(?u, ??r^{'})$

Other potential constraints can also be defined similarly.

In the following, we present a simple example of entities and policy rules that can be defined using the proposed language. There are three different CSPs namely *ProviderA*, *ProviderB* and *ProviderC* each of them offers two services and own some objects.

Roles

RoleA a sbpsl:Role, RoleB a sbpsl:Role,

RoleC a sbpsl:Role

Subjects

SubjectA a sbpsl:Subject

hasRole RoleA

isAssociatedWithProvider ProviderA,

SubjectB a sbpsl:Subject

hasRole RoleB

isAssociatedWithProvider ProviderB,

SubjectC a sbpsl:Subject

hasRole RoleC

isAssociatedWithProvider ProviderC

Objects

ObjectA a sbpsl:Object

isAssociatedWithService ServiceA.1

isOwnedByProvider ProviderA,

ObjectB a sbpsl:Object

isAssociatedWithService ServiceB.1

isOwnedByProvider ProviderB,

ObjectC a sbpsl:Object

isAssociatedWithService ServiceC.1

isOwnedByProvider ProviderC

Actions

Read a sbpsl:Action, Write a sbpsl:Action,

Execute a sbpsl:Action

Provider

ProviderA a sbpsl:Provider, ProviderB a sbpsl:Provider,

ProviderC a sbpsl:Provider

Service

ServiceA.1 a sbpsl:Service offeredBy ProviderA,

ServiceA.2 a sbpsl:Service offeredBy ProviderA,

ServiceB.1 a sbpsl:Service offeredBy ProviderB,

ServiceB.2 a sbpsl:Service offeredBy ProviderB,

ServiceC.1 a sbpsl:Service offeredBy ProviderC,

ServiceC.2 a sbpsl:Service offeredBy ProviderC

Now that the classes have been instantiated, we can define policy rules such as *[ProviderA, SubjectB, ObjectA, Read, ServiceA.1]* which is interpreted as *ProviderA* permits *SubjectB* to perform *Read* action on *ObjectA* which is related to *ServiceA.1*. If we define *ObjectA*.1

$isDescendantOf\ ObjectA$, the above rule applies to $ObjectA.1$ too and *SubjectB* can perform *Read* action on *ObjectA.1* in addition to *ObjectA*.

4.1.3.2 Multi-Cloud Collaboration In this section, we show how the proposed ontology can facilitate collaboration among multiple CSPs. We describe multiple collaboration scenarios and show how to extend the proposed ontology to handle each scenario. In these scenarios, we assume that the CSPs support RBAC.

Consider the following scenario: Alice who is a user of CSP CSP_2 and has a resource called $HolidayPics$ on the CSP_2's $PhotoSharingSrv$ service, wants to specify a policy that allows her family members access to this resource. Bob who is a family member of Alice, however, is a user of another CSP, CSP_1 and wants to access the $HolidayPics$ which is stored on the CSP_2 and edit some of the photos using CSP_1's $PhotoEditingSrv$ service.

These elements are modeled in the knowledge base of the SBPMS as follows.

- CSP_1 and CSP_2 are instances of the provider.
- Alice and Bob are instances of the subject.
- $FamilyMember$ is an instance of the role.
- $HolidayPics$ is an instance of the object.
- Read and Edit are instances of the action.
- $PhotoSharingSrv$ and $PhotoEditingSrv$ are instances of the service.

The policy is defined as $[Any, FamilyMember, HolidayPics, Read, PhotoEditingSrv]$ meaning that any user who is assigned to the role $FamilyMember$ is allowed to read the $HolidayPics$ which is associated with the $PhotoSharingSrv$ service.

This policy is straightforward for users of the CSP_2 who are assigned to the $FamilyMember$ role, since the $HolidayPics$ is stored on the CSP_2. However, users of other CSPs who are $FamilyMember$ of Alice also should be given access to the $HolidayPics$.

We take CSP_1 as an example here and assume that Bob who is a user of the CSP_1 and a $FamilyMember$ of Alice wants to access the $HolidayPics$ on the CSP_2. In order to do this, CSP_1 and CSP_2 should collaborate with each other. The process is as follows:

- To make resources accessible by users from collaborating CSPs, CSPs CSP_1 and CSP_2 come to an agreement regarding the sharing of the resources.
- A contract is established for CSP_1 to access CSP_2's resources.
- CSP_1 gets the resource description from the knowledge base of the SBPMS.
- CSP_2 creates a "virtual user", $CollaboratorCSP_1$, to represent authorized users from CSP_1 for future resources invocations.
- CSP_2 associates $CollaboratorCSP_1$ to a role with appropriate permissions, $FamilyMember$.

Note that details of agreement and contract establishment is beyond scope of this dissertation; some existing work in the literature can be used for this purpose [40, 41].

The policy is in the SBPMS and since the resource in the rule is related to the CSP_2, other CSPs cannot see the rule for privacy reasons. The SBPMS should not by default share information about resources with CSPs other the one that the resource is stored on. The data owner may not want to allow other CSPs to see information about resources stored on one specific CSP. In order to collaborate, however, there needs to be a way to share information about resources which could be accessible to collaborating parties.

In order to preserve privacy of resources, data owners can determine via defined policies which CSPs can have access to which part of the knowledge base. Once the policies are defined by data owners, the SBPMS determines which CSPs may be allowed to access the resources and information about those resources is shared with corresponding CSPs. When the contract is established between the CSP_1 and the CSP_2, the SBPMS shares policies of the associated resources with the CSP_1 too. In the above example, after Alice defines the policy on the $HolidayPics$, the SBPMS using its knowledge base can determine that CSP_1 has users who are $FamilyMember$ of Alice and share information about the $HolidayPics$ with the CSP_1 too.

Once the CSPs have established a sharing agreement, the following process is carried out at runtime:

- CSP_1 delivers a ticket to its user, Bob, proving that he is authorized to access CSP_2's resource.
- In CSP_2's side, Bob presents the ticket proving that he is authorized by CSP_1.

- CSP_2 verifies the contract rule and assigns the virtual user $CollaboratorCSP_1$ to Bob and he will thus be considered as an internal user having internal permissions.

In this way, the same policy is used for both internal and external users.

The above scenario is when the access request is made at the CSP where the resource is stored on and policy decision making is done locally at the CSP. Another scenario is when policy decisions are made at the SBPMS and only enforcement is done by the CSP. In this case, the access request is sent to the SBPMS and it has all the information about resources, so there is no need for CSP_1 and CSP_2 to negotiate for sharing resources. However, we still need the virtual user at the CSP_2 to represent authorized users from CSP_1. Also there is no need for Bob to get a ticket from CSP_1 and present to CSP_2 as the SBPMS has all the required information and can decide whether Bob is authorized according to the policy. The process is as follows:

- A contract is established for CSP_1 to access CSP_2's resources.
- CSP_2 creates a "virtual user", $CollaboratorCSP_1$, to represent authorized users from CSP_1 for future resources invocations.
- CSP_2 associates $CollaboratorCSP_1$ to a role with appropriate permissions, $FamilyMember$.
- CSP_2 assigns the virtual user $CollaboratorCSP_1$ to Bob and he will thus be considered as an internal user having internal permissions.

We extend the ontology to facilitate interoperation between CSPs and show how to map information in order to enable interoperation.

Assuming RBAC is access control mechanism of choice, in addition to inheritance relationships among roles which are modeled using $subRoleOf$ property, we introduce another relationship for mapping between roles of difference providers. We also define the following properties to show state of a role. These are modeled using attributes for a role and *EnabledRole* indicates that subjects/ users who are authorized for the role can activate the role, *DisabledRole* indicates that the role can not be activated in a session, while *ActivatedRole* indicates that there is at least one user who has activated the role.

In the ontology, the *Provider* determines which CSP domain the role belongs to; we add the property $isAssocitedWithProvider$ to the class *Role* to show domain of the role. We

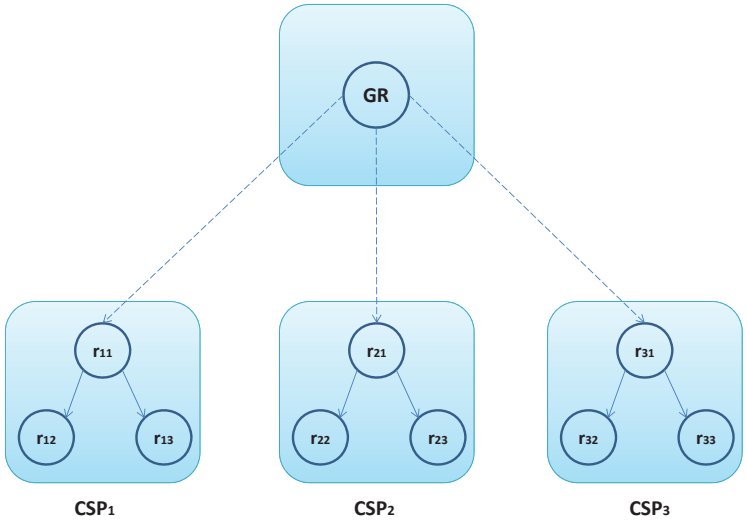


Figure 7: The Collaboration Scenario With Mediator

also add the property *RoleMappingEnabled* to the class *Role* and use it to model the role mapping relationships.

In the first scenario, there is an entity that mediates accesses to individual systems through a global policy. In this situation, the access control and interoperation needs are typically predefined, and a global policy is created by integrating all the individual policies to facilitate such interoperation needs.

As shown in the Figure 7, the global roles are mapped to the local roles using *RoleMappingEnabled* property. The global roles are stored in the SBPMS and the local roles on the CSPs. The solid lines indicate inheritance relationship among roles where the dashed lined describe role mapping among various CSPs.

We define the followings to map the global role to the local roles and describe the semantics for interoperation. The SBPMS is defined as a global domain to contain global roles and is an instance of the *Provider*; the *GR* is defined as an instance of the *Role*. The followings descibe the mapping relationships.

GR isAssocitedWithProvider SBPMS

GR RoleMappingEnabled r_{11}

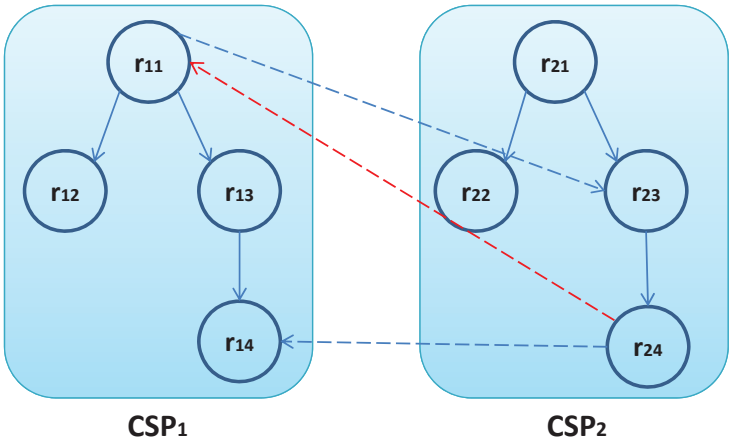


Figure 8: The Dynamic Collaboration Scenario

GR RoleMappingEnabled r_{21}

GR RoleMappingEnabled r_{31}

Note that in this case, the SBPMS can act as mediator and manage collaboration requirements among multiple CSPs. As mentioned before, the global roles are generated and stored in the SBPMS; then the SBPMS exports appropriate local roles into the CSPs to facilitate collaboration. Alternatively, the collaboration can be done among multiple instances of the SBPMS. For example, if different organizations deploy their own instances of the SBPMS and then want to collaborate, one of the SBPMSs can act as mediator and facilitate the collaboration process.

In the second scenario, there is no mediator entity and the collaborator CSPs dynamically come together to share information for a period of time. The role mapping relationships can be defined similar to the previous scenario. However, in such interoperation scenarios, the role mapping relationships may introduce a cycle that enables a subject lower in the access control hierarchy to acquire the permissions of a subject that is higher in the hierarchy [45]. For example, as shown in Figure 8 if we add the red dashed line (i.e., edge between r_{24} and r_{11}) as a new interoperation link, users that are originally authorized for role r_{24} and not for

role r_{23}, now can get authorized for the role r_{23} through the role mapping relationships from role r_{24} to role r_{11} to role r_{23}.

In order to prevent these situations, we must define rules to restrict property $RoleMappingEnabled$ and prevent inheritance cyclic conflicts [45]. For example, in Figure 8 the red dashed line mapping between role r_{24} and role r_{11} cannot be established.

The following three situations are cases where cyclic conflicts can happen if we add a new role mapping relationship between role r_1 and role r_2:

- A role mapping relationship already exists between junior role of r_1 and senior role of r_2.
- A role mapping relationship already exists between junior role of r_1 and role r_2.
- A role mapping relationship already exists between role r_1 and senior role of r_2.

The following rules can be used to capture each of these cases [45].

- $subRole(?r_1, ?r^{'}) \wedge subRole(?r^{"}, ?r_2) \wedge RoleMappingEnabled(?r^{'}, ?r^{"})$
  $\rightarrow \neg RoleMappingEnabled(?r_2, ?r_1)$
- $subRole(?r_1, ?r^{'}) \wedge RoleMappingEnabled(?r^{'}, ?r_2) \rightarrow \neg RoleMappingEnabled(?r_2, ?r_1)$
- $subRole(?r^{"}, ?r_2) \wedge RoleMappingEnabled(?r_1, ?r^{"}) \rightarrow \neg RoleMappingEnabled(?r_2, ?r_1)$

Similar to the case with mediator, collaboration could be done among multiple CSPs or multiple SBPMSs. The process is same in both cases; the parties can come together dynamically and collaborate based on their needs. Secure interoperation such as these have complexity issues, which is outside the scope of this dissertation. We refer the interested readers to [43].

4.1.3.3 OWL-GTRBAC: Specification and Enforcement of Temporal Constraints using OWL

In this section, we show how to model temporal constraints and restrictions in the Generalized Temporal Role Based Access Control (GTRBAC) model using OWL. We first provide an overview of the GTRBAC model and its temporal constraints; Then we define OWL ontologies that represent these temporal constraints in GTRBAC.

Overview: GTRBAC

The GTRBAC model is an extension of Temporal RBAC (TRBAC) which is an RBAC extension to address temporal constraints [26]. In particular, GTRBAC model allows one to

express periodic and duration constraints on roles, user-role assignments, and role-permission assignments. Numerous activation constraints including cardinality constraints and maximum active duration constraints can lead to further restriction of activation of a role in an interval. The GTRBAC model extends the structure of the TRBAC model such that its features including event and trigger expressions subsume those of TRBAC.

Temporal Constraints in GTRBAC: In GTRBAC, a role can have one of the three states: disabled, enabled, and active. Being in disabled state means that a user cannot acquire the permissions associated with the role. A role in the disabled state can be enabled. When a role is in enabled state, users who are authorized to use the role at the time of the request can activate the role. If a user activates a role in enabled state, the state of the role becomes active. The active state indicates that there is at least one user who has activated the role. If a disabling event occurs, roles in the enabled or active state transit to the disabled state. The model allows the specification of the following types of constraints: temporal constraints on role enabling, user-role, and role-permission assignments, activation constraints, runtime events, constraint enabling expressions, and triggers. Priorities are associated with each event in GTRBAC. $(Prios, \preceq)$ is defined as a totally ordered set of priorities. In GTRBAC, event expressions, priorities, and status predicates are used to express the constraints.

Periodicity and Duration Constraints on Role Enabling and Assignments: The model uses periodicity constraints to specify the intervals. Periodicity constraints are intervals during which a role can be enabled or disabled, and during which a user-role assignment or a role permission assignment is valid. Generally, periodicity constraint expressions are specified by $(I, P, pr : E)$. The pair (I, P) specifies the intervals during which an event E takes place. E can be one of the assignment events: "$assign_p/deassign_p$ p to r" or "$assign_u/deassign_u$ u to r" or a role enabling event: "enable/disable r". pr indicates the priority of event.

The model uses duration constraints to specify durations for which enabling or assignment of a role is valid. In case of an event occurrence, the duration constraint associated with the event validates the event for the specified duration only. Generally, the duration constraint expressions for role enabling and assignment are specified by $([(I, P)|D], D_x, pr : E)$,

Table 1: GTRBAC Example: Access Policy for Medical Information System

1	a	(DayTime, enable DayDoctor), (NightTime, enable NightDoctor)
	b	((M, W, F), $assign_u$, Adams to DayDoctor), ((T, Th, S, Su), $assign_u$, Bill to DayDoctor)
	c	(Everyday between 10am-3pm, $assign_u$, Carol to DayDoctor)
2	a	($assign_u$ Ami to NurseInTraining), ($assign_u$ Elizabeth to DayNurse)
	b	c1=(6 hours, 2 hours, enable NurseInTraining)
3	a	(enable DayNurse $\rightarrow$ enable c1)
	b	(activate DayNurse for Elizabeth $\rightarrow$ enable NurseInTraining after 10 min)
	c	(enable NightDoctor $\rightarrow$ enable NightNurse after 10 min), (disable NightDoctor $\rightarrow$ disable NightNurse after 10 min)

where x is either R, U, or P, corresponding to events: "enable/disable r," "$assign_u/deassign_u$ r to u," and "$assign_p/deassign_p$ p to r," respectively. D and D_x refer to the durations such that $D \leq D_x$. The symbol "—" between (I, P) and D indicates that either (I, P) or D is specified. The square bracket in $[(I, P)|D]$ implies that this parameter is optional.

Temporal Constraints on Role Activation: Duration constraints can be applied on role activations whereas periodicity constraints on role activations should not be applied. The duration constraints can be classified into two types: total active duration constraint and maximum duration per activation constraint. The total active duration constraint on a role restricts the number of the role's activation duration in a given period to a specified value. The total active duration can be specified on per-role and per-user-role basis. Per-role constraint restricts the total active duration for a role, while per-user-role constraint restricts the total active duration for a role by a particular user.

Example Scenario:

To illustrate our approach we use a simple scenario from a medical information system that is adopted from [98]. The example is shown in Table 1. In row 1a, the enabling times of

DayDoctor and NightDoctor roles are specified as a periodicity constraint. The (I, P) forms for DayTime (9:00 a.m.-9:00 p.m.) and NightTime (9:00 p.m.-9:00 a.m.) are as follows:

$DayTime = ([12/1/2008, \infty], all.Days + 10.Hours \triangleright 12.Hours)$ and

$NightTime = ([12/1/2008, \infty], all.Days + 22.Hours \triangleright 12.Hours$

In constraint 1b, Adams is assigned to the role of DayDoctor on Mondays, Wednesdays, and Fridays, whereas Bill is assigned to this role on Tuesdays, Thursdays, Saturdays, and Sundays. The assignment in constraint 1c indicates that Carol can assume the DayDoctor role everyday between 10:00 a.m. and 3:00 p.m. Constraint 2b specifies a duration constraint of 2 hours for the enabling time of the NurseInTraining role, but this constraint is valid only for 6 hours after the constraint c1 is enabled. Consequently, once the NurseInTraining role is enabled, Ami can activate the NurseInTraining role at the most for two hours.

Trigger 3a indicates that the constraint c1 in row 2b is enabled once the DayNurse is enabled. As a result, the NurseInTraining role can be enabled within 6 hours. Trigger 3b indicates that 10 min after Elizabeth activates the DayNurse role, the NurseInTraining role is enabled for a period of 2 hours. As a result, a nurse-in-training can have access to the system only if Elizabeth is present in the system. In other words, once the roles are assumed, Elizabeth acts as a training supervisor for a nurse-in-training. Note that Elizabeth can activate the DayNurse role multiple times within a duration of 6 hours after the DayNurse role is enabled.

Temporal Constraints in OWL

In this section we describe OWL ontology that conceptualize temporal constraints introduced in GTRBAC. The ontology defines restrictions and constraints in GTRBAC including activation constraints, cardinality constraints, and temporal constraints as shown in the Figure 9.

We create classes representing basic RBAC components (i.e., user, role, etc) as follows. The users are modeled as an OWL class **User**. The instances of this class represent the users/subjects on which the policies are defined. The association between a user and a role is defined by the ObjectProperty *hasRole(User, Role)*. Moreover, we can link a user to a permission by the ObjectProperty *hasPermission (User, Permission)*.

User rdfs:subClassOf owl:Thing

Figure 9: The GTRBAC Ontology

hasRole a rdfs:Property, owl:ObjectProperty

rdfs:domain :User;

rdfs:range :Role.

hasPermission a rdfs:Property, owl:ObjectProperty

rdfs:domain :User;

rdfs:range :Permission.

We define three classes Action, Object and Permission to represent actions, objects/resources and permissions respectively. The Permission class binds a user to an action-object using two properties hasAction and hasObject.

Action rdfs:subClassOf owl:Thing

Object rdfs:subClassOf owl:Thing

Permission rdfs:subClassOf owl:Thing

hasAction a rdfs:Property, owl:FunctionalProperty

rdfs:domain :Permission;

rdfs:range :Action.

hasObject a rdfs:Property, owl:FunctionalProperty

rdfs:domain :Permission;

rdfs:range :Object.

In order to represent a role hierarchy $\langle R, \preceq \rangle$, we model roles as an OWL class **Role**, and all the roles in R as instances of this class. The $\preceq$ relation is represent by the OWL property *subRoleOf(Role, Role)*. Furthermore, we define subRoleOf to be transitive by making it an instance of owl:TransitiveProperty class. In this way, a reasoner can infer that if $subRoleOf(R_i, R_j)$ and $subRoleOf(R_j, R_k)$, then $subRoleOf(R_i, R_k)$. For example, we can model the relation between Professor and AssistantProfessor by adding to the ontology the property subRoleOf(Assistant Professor, Professor). The association between a role and a permission can be defined by ObjectProperty *hasPermission(Role, Permission)*.

Role rdfs:subClassOf owl:Thing

subRoleOf a rdfs:Property, owl:TransitiveProperty

rdfs:domain :Role;

rdfs:range :Role.

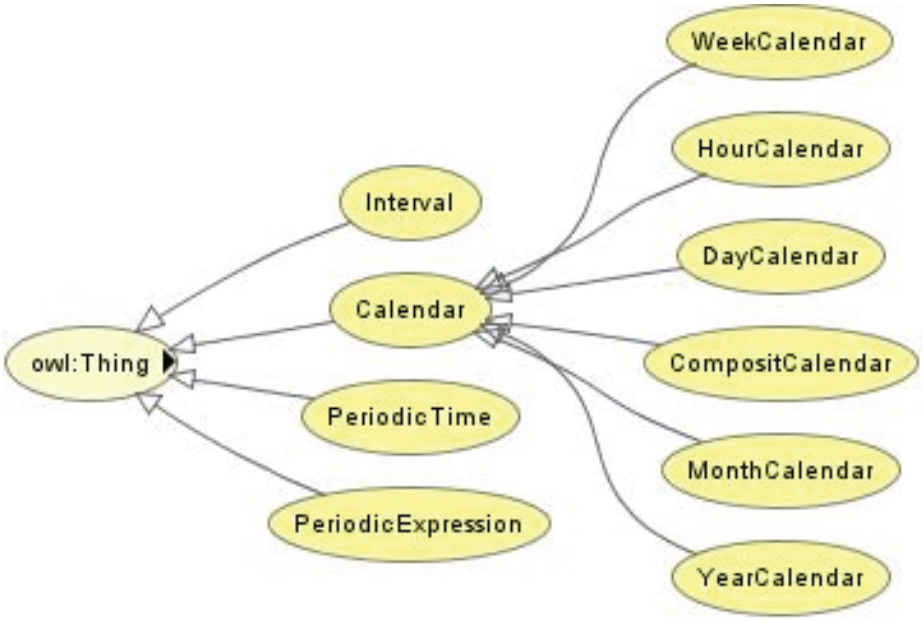


Figure 10: The Periodic Expressions in GTRBAC

hasPermission a rdfs:Property, owl:ObjectProperty

rdfs:domain :Role;

rdfs:range :Permission.

The rest of this section describes the ontology which conceptualizes temporal constraints defined in the GTRBAC model.

Periodic Expressions: Periodic expressions are the basis for representing temporal constraints and shown in Figure 10. Calendar classes represent temporal units (i.e. year, month, week, day, and hour). Using the CompositeCalendar, it is possible to define temporal expression based on different temporal units.

YearCalendar rdfs:subClassOf Calendar

MonthCalendar rdfs:subClassOf Calendar

WeekCalendar rdfs:subClassOf Calendar

DayCalendar rdfs:subClassOf Calendar

HourCalendar rdfs:subClassOf Calendar

CompositCalendar rdfs:subClassOf Calendar

year a rdfs:Property, owl:ObjectProperty

rdfs:domain : CompositCalendar;

rdfs:range : YearCalendar.

month a rdfs:Property, owl:ObjectProperty

rdfs:domain : CompositCalendar;

rdfs:range : MonthCalendar.

week a rdfs:Property, owl:ObjectProperty

rdfs:domain : CompositCalendar;

rdfs:range : WeekCalendar.

day a rdfs:Property, owl:ObjectProperty

rdfs:domain : CompositCalendar;

rdfs:range : DayCalendar.

hour a rdfs:Property, owl:ObjectProperty

rdfs:domain : CompositCalendar;

rdfs:range : HourCalendar.

Interval class is defined with two properties, beginTime and endTime as follows.

Interval rdfs:subClassOf owl:Thing

beginTime a rdfs:Property, owl:FunctionalProperty

rdfs:domain : Interval;

rdfs:range :date.

endTime a rdfs:Property, owl:FunctionalProperty

rdfs:domain : Interval;

rdfs:range : date.

The PeridociExpression class represents duration, for example 10 hours from 2009-06-09, 10.am.

PeriodicExpression rdfs:subClassOf owl:Thing

baseCalendar a rdfs:Property, owl:ObjectProperty

rdfs:domain : PeriodicExpression;

rdfs:range : CompositCalendar.

durationUnit a rdfs:Property, owl:DataTypeProperty

rdfs:domain : PeriodicExpression;

rdfs:range : String, owl:DataRange:{Year, Month, Week, Day, Hour}.

durationVariable a rdfs:Property, owl:FunctionalProperty

rdfs:domain : PeriodicExpression;

rdfs:range : Integer.

The PeriodicTime class is conceptualized by hasInterval and hasPeridociExpression properties.

PeriodicTime rdfs:subClassOf owl:Thing

hasInterval a rdfs:Property, owl:ObjectProperty

rdfs:domain : PeriodicTime;

rdfs:range : Interval.

hasPeriodicExpression a rdfs:Property, owl:ObjectProperty

rdfs:domain : PeriodicTime;

rdfs:range : PeriodicExpression.

Events in GTRBAC: In this section, we represent various types of events defined in the GTRBAC model as shown in Figure 11. Each Event class has properties related to role, user, or permission depending on the type of relation being represented.

Event rdfs:subClassOf owl:Thing

UserRoleAssignment rdfs:subClassOf Event

hasUser a rdfs:Property, owl:ObjectProperty

rdfs:domain : UserRoleAssignment;

rdfs:range : User.

hasRole a rdfs:Property, owl:ObjectProperty

rdfs:domain : UserRoleAssignment;

rdfs:range : Role.

Similarly, we define other events such as **UserRoleDeassignment**, **RolePermissionAssignment**, **RolePermissionDeassignment**, **RoleEnabling**, **RoleDisabling**, **RoleActivation**, **ConstraintEnabling**, and **ConstraintDisabling**.

The Trigger class conceptualizes a trigger in GTRBAC. The class has triggeringEvent and triggeredEvent properties.

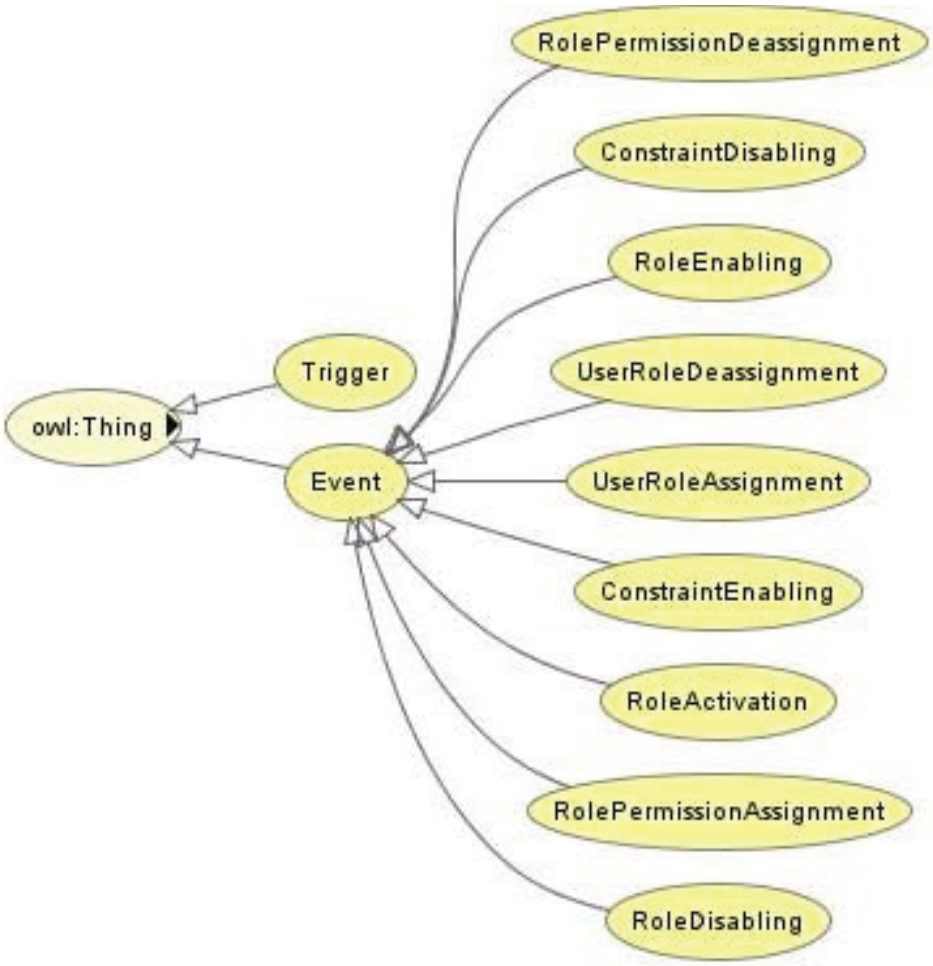


Figure 11: The GTRBAC Events

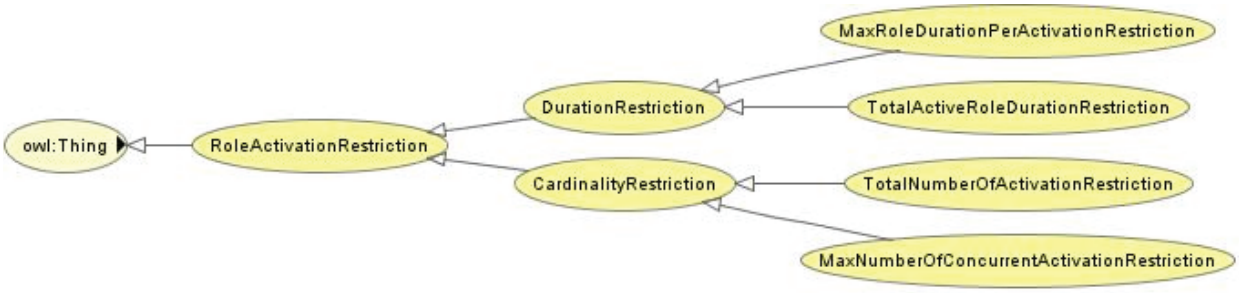


Figure 12: The GTRBAC Restrictions

Trigger rdfs:subClassOf owl:Thing

triggeringEvent a rdfs:property, owl:objectProperty

rdfs:domain Trigger

rdfs:range Event

triggeredEvent a rdfs:property, owl:objectProperty

rdfs:domain Trigger

rdfs:range Event

Restrictions and Constraints: The restrictions and constraints, which are most important concepts in GTRBAC, represent temporal constraints in different ways as shown

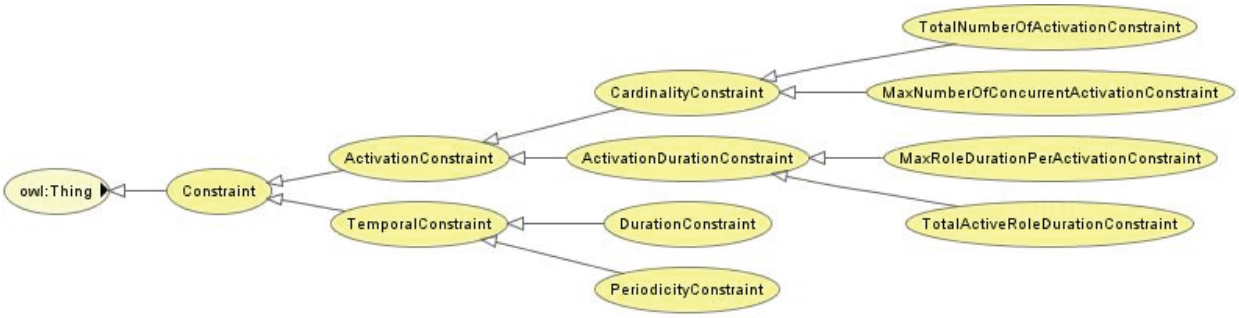


Figure 13: The GTRBAC Constraints

in Figure 12 and Figure 13 respectively. While there are several types of restrictions in GTRBAC, we describe only role activation restrictions. The RoleActivationRestriction class associates with RoleActivation and RoleDeactivation events.

RoleActivationRestriction rdfs:subClassOf owl:Thing

associatedWith a rdfs:Property, owl:ObjectProperty

rdfs:domain : RoleActivationRestriction;

rdfs:range : Event, owl:allValuesFrom: RoleActivation.

user a rdfs:Property, owl:ObjectProperty

rdfs:domain : RoleActivationRestriction;

rdfs:range : User, owl:maxCardinality:1.

restrictTo a rdfs:Property, owl:FunctionalProperty

rdfs:domain : RoleActivationRestriction;

rdfs:range : String, owl:DataRange:{PERROLE, PERUSERROLE}.

It has two subclasses according to the type of restriction; **CardinalityRestriction** and **DurationRestriction** class. The former restricts role activation by using activeCardinality property, while the latter uses activeDuration property.

The CardinalityRestriction is represented by two properties: activeCardinality and defaultCardinality. The **MaxNumberOfConcurrentActivationRestriction** and **TotalNumberOfActivationRestriction** are defined as subclasses of CardinalityRestriction.

The **DurationRestriction** is represented by activateDuration property. The **MaxRoleDurationPerActivationRestriction** and **TotalActiveRoleDurationRestriction** are defined as subclasses of DurationRestriction.

The Constraint classes represent temporal policies in GTRBAC. Each Constraint class is associated with only one event. The ActivationConstraint class is related to activation events and it is described by associatedWith, canDuration, and restriction properties.

Constraint rdfs:subClassOf owl:Thing

ActivationConstraint rdfs:subClassOf Constraint

associatedWith a rdfs:Property, owl:ObjectProperty

rdfs:domain : Constraint;

rdfs:range : Event, owl:allValuesFrom: RoleActivation.

canDuration a rdfs:Property, owl:ObjectProperty

rdfs:domain : ActivationConstraint;

rdfs:range : PeriodicTime.

restriction a rdfs:Property, owl:ObjectProperty

rdfs:domain : ActivationConstraint;

rdfs:range : RoleActivationRestriction.

It has two subclasses according to restriction type (duration or cardinality), **ActivationDurationConstraint** and **CardinalityConstraint** which are described by restriction property. The **MaxRoleDurationPerActivationConstraint** and **TotalActiveRoleDurationConstraint** are subclasses of ActivationDurationConstraint and the **MaxNumberOfConcurrentActivationConstraint** and **TotalNumberOfActivationConstraint** are subclasses of CardinalityConstraint. They are distinguished by owl:allValuesFrom on restriction property.

The TemporalConstraint classes are related to all events except (de)activation events.

TemporalConstraint rdfs:subClassOf Constraint

associatedWith a rdfs:Property, owl:ObjectProperty

rdfs:domain : Constraint;

rdfs:range : Event, owl:allValuesFrom: {UserRoleAssignment, UserRoleDeassignment, RoleEnabling, RoleDisabling, RolePermissionAssignment, RolePermissionDeassignment}.

The class is divided into two subclasses according to the type of restriction, **DurationConstraint** and **PeriodicityConstraint**. DurationConstraint class has two additional properties, canDuration and eventDuration. PeriodicityConstraint has periodicTime property in order to represent periodic restrictions.

Enforcing Restrictions and Constraints

In this section, we apply the presented model to our example scenario. The followings are policies shown in table 1. The specification in illustrative not exhaustive given space constraints.

Users

Adams a gtrbac:User, Bill a gtrbac:User

Roles

DayDoctor a gtrbac:Role, NightDoctor a gtrbac:Role,

NurseInTraining a gtrbac:Role

Events

EnableDayDoctor a gtrbac:RoleEnabling

EnableNurseInTraining a gtrbac:RoleEnabling

AdamsToDayDoctor a gtrbac:UserRoleAssignment

EnableTable1-2b a gtrbac:ConstraintEnabling

Intervals

Interval1 a gtrbac:Interval

beginTime 20031201

endTime 99991231

Periodic Times

DayTime a gtrbac:PeridicTime

interval Interval1

hasPeriodicExpression DayTimeExp

DayTimeExp a gtrbac:PeriodicExpression

durationUnit HOUR

durationValue 12

baseCalendar {AllYears, AllMonths, AllWeeks, AllDays, 10}

Table1-1b-1-time a gtrbac:PeridicTime

interval Interval1

hasPeridocExpression Table1-1b-1exp

Table1-1b-1exp a gtrbac:PeriodicExpression

durationUnit HOUR

durationValue 24

baseCalendar {AllYears, AllMonths, AllWeeks,

(Monday, Wednesday, Friday), AllHours}

It is similar for Table1-1b-2-time and Table1-1c-time.

Constraints for Row1

Table1-1a-(a) a gtrbac:PeriodicityConstraint

associatedWith EnableDayDoctor

periodicTime DayTime

Table1-1b-(a) a gtrbac:PeriodcityConstraint

associatedWith AssignAdamsToDayDoctor

periodicTime Table1-1b-1-time

Constraints in Row2

Table1-2b a gtrbac:DurationConstratin

associatedWith EnableNurseInTraining

canDuration 6, HOUR

eventDuration 2, HOUR

Triggers in Row3

Table1-3b a gtrbac:Trigger

triggeringEvent ActivateDayNurse

triggeringPostfix for, Elizabeth

triggeredEvent EnableNurseInTraingin

triggeredPostfix after, 10, MINUTE

Table1-3c-(a) a gtrbac:Trigger

triggeringEvent EnableNightDoctor

triggeredEvent EnableNightNurse

triggeredPostfix after, 10, MINUTE

Now that we have introduced an ontology to represent entities in the system and modeled GTRBAC using OWL, we show how the policy management process works. In order to show expressiveness power of the the proposed ontology, we have shown that how it can be used to represent various models such as RBAC, constraints in RBAC, temporal constraints of GTRBAC, etc.

One can specify policies in various models using OWL ontologies. We can specify the policies using the general semantic based policy specification introduced in section 4.1.3.1

which is compatible with XACML. The temporal constraint and GTRBAC policies could be specified using ontologies described in section 4.1.3.3. The policies can also be specified in RBAC using our proposed ontology or by employing existing approaches such as the ROWLBAC model proposed by *Finin et al.* [99, 109]. Regardless of the model used for specification, the policies will be transferred to RBAC policies and stored at the global RBAC policy base. In order to do this, if GTRBAC is the model of choice, we can use the approach proposed by *Zhang et al.* to convert GTRBAC policies into RBAC policies [42]. If we use XACML-compatible specification, we can use RBAC profile of XACML to convert policies to RBAC policies [44].

Although in general the PMaaS and the SBPMS, which is an instance of the PMaaS, do not impose any restriction on the policy specification, for simplicity purpose we assume that the CSPs use RBAC model. All the policies are transferred to RBAC policies and stored at the global RBAC policy base. Once the policies are at the global RBAC policy base, they will be exported to associated local policy RBAC bases which in turn used by their respective RBAC engines to make authorization decisions and enforce them. Note that the access requesters in this case are sent to the CSP and policy decision and enforcement processes are handled locally by the CSPs and only policy specification is done at the PMS.

Alternatively, we can perform decision making at the PMS side too. In this case, we don't need to keep local RBAC policy bases at the CSPs; The access request are sent to the PMS rather than individual CSPs, decisions are made at the PMS based on policies on the global RBAC policy base, and the appropriate authorization decisions are communicated to the RBAC engines of corresponding CSPs to enforce the decision.

4.1.4 Proof of Concept Implementation and Experimental Results

In this section, we explain a proof of concept implementation of our proposed system and present our performance evaluation results.

Our proof of concept implementation includes developing a Java library to abstract the proposed framework. The library provides an interface which enables the cloud users to manage the ontologies, and specify the authorization policies. Our implementation contains

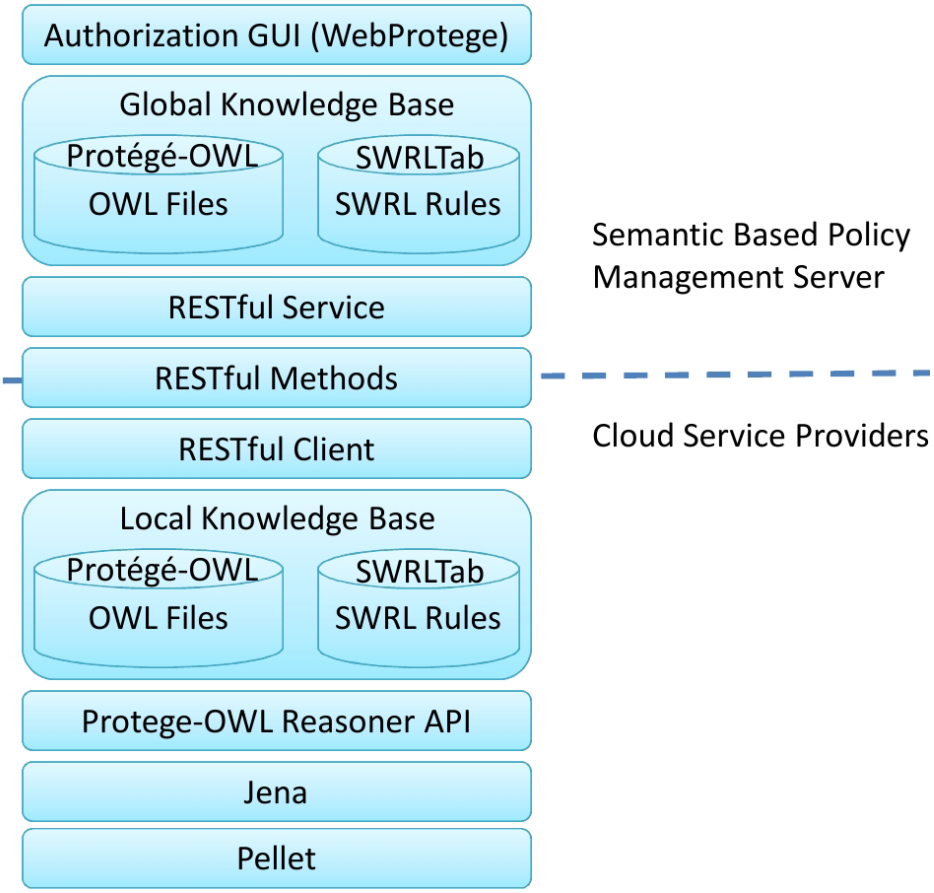


Figure 14: The Implementation Architecture of the Proposed Framework

two different Java based components: the CSP and the SBPMS. The SBPMS contains authorization API, knowledge management and authorization GUI and the CSP contains the provider authorization API, knowledge management and semantic based PDP.

As shown in Figure 14, the authorization API and the provider authorization API are published using RESTful Web service technology to enable remote invocation of the methods for the different entities involved. The reason we chose RESTful Web service technology is that many cloud providers make use of RESTful architecture to provide RESTful APIs to "execute services which can seamlessly scale and have modular architecture, ease of integration and extensibility" [59]. Cloud computing services by nature are distributed and use of web-based RESTful APIs by cloud customers is a logical solution for cloud services [60]. A RESTful Web service is implemented in the SBPMS side and the CSPs are implemented as RESTful clients. The provider authorization API establishes a secure channel to the semantic based policy management server using *SSL*. Authentication of the provider and server is done using *OAuth* which is an open protocol to allow secure API authorization in a simple and standard method from web applications and a simple way to publish and interact with protected data [55]. The authorization API provides methods to *insert*, *update*, *remove*, *search* and *access* information in the knowledge base.

Our proposed framework uses *Protege* which offers an optimized and efficient solution to manage large knowledge bases with authorization information [54]. *Protege* is a Java tool that provides an extensible architecture and a suite of tools for the creation of customized knowledge-based applications with ontologies. We use *Protege-OWL* (http://protege.stanford.edu/overview/protege-owl.html) that is an editor to enable users to build ontologies for the Semantic Web, in particular in the OWL. It implements a rich set of knowledge-modeling structures and actions that support the creation, visualization, and manipulation of ontologies in various representation formats. For SWRL rules, our implementation makes use of SWRLTab (http://protege.cim3.net/cgi-bin/wiki.pl?SWRLTab) which is a development environment for working with SWRL rules in Protege-OWL. It supports the editing and execution of SWRL rules. Further, Protege can be extended by way of a plug-in architecture and a Java-based API for building knowledge-based tools and applications.

In order to reason over the ontologies represented in Protege-OWL, we use *Jena* [103] and *Pellet* [53]. *Jena* is used as a Java API to manage ontologies and *Pellet* is used as DL reasoner. The former is the standard Java library for ontology management while the latter offers high expressiveness power when dealing with OWL 2 ontologies and performs incremental consistency checking. Our implementation uses the *Protege-OWL Reasoner API* (http://protegewiki.stanford.edu/wiki/ProtegeReasoner
API) that provides programmatic access to a reasoner. It provides methods for consistency checking, classification, etc. of an ontology as well as methods for getting the inferred information for a particular OWL entity. The *ProtegePelletJenaReasoner* implementation converts a *Protege-OWL* model into a *Jena* model and then it uses the existing *Pellet* reasoner connection available in *Jena* for the inference.

The authorization GUI is a graphical Web user interface that enables cloud users to access authorization services in a usable way. Our implementation uses *WebProtege* (http://protegewiki.stanford.edu/wiki/WebProtege) which is a lightweight, web-based ontology editor developed to support the process of ontology development in a web environment. The cloud users can use this interface to search the knowledge base, specify authorization policies and request authorization proofs.

We used RESTful Web service technology because it allowed us to make use of existing components and put them together to implement the proof of concept prototype and perform the simulations. A functioning system, however, needs to take into account features and functionalities that are required for real world deployment. Actual deployment of the proposed framework in practice is beyond scope of this dissertation.

The framework is developed to be used by cloud customers, both individual and enterprize customers. As mentioned before, it can be deployed as public cloud service or a private cloud by enterprize. The eventual beneficiary of the proposed framework are cloud customers and it enables users to manage access policies to their resources.

After implementing our proof of concept prototype based on the architecture discussed above, we present its performance evaluation. We used two different machines to do our experiments: a server and a client.

The server is an Intel Core i5-520M 2.40 GHz with 8 Gbytes RAM and Windows 7

Ultimate running the SBPMS server. The client is an AMD Opteron 252 2.60 GHz with 8 Gbytes RAM and Windows 7 Ultimate running an application that simulates CSPs that utilize the SBPMS server using the authorization client API. The SBPMS is launched as a web service and the simulator starts 100 threads each representing a CSP using the policy management system. All the threads are run in parallel to stress the policy management system with simultaneous invocations.

Ontology and Policy Generation Process

Since we did not have any real data, we simulated OWL files and SWRL rules. We generated 100 separate OWL files, one for each CSP and 200 SWRL rules using those OWL files. In order to do this, we used the semantic based specification introduced in section 4.1.3 based on the meta-model shown in Figure 6.

In the data simulation process, we defined all entities in OWL ontology, their properties, relationships, etc. We generated instances of these classes including 100 providers, 500 subjects, 2000 objects, 8 actions and between 1 and 10 services for each CSP. These services are randomly assigned to the CSPs and each of the 100 providers has at least 1 and at most 10 services. The actions we used in data generation process are *read*, *write*, *execute*, *parallel write*, *parallel read*, *parallel execute*, *parallel read and write*, *parallel read and execution*. This data was generated and stored at 100 separate OWL files each associated with one CSP. Note that these OWL files are not completely different and there are some similarities among them as in real world scenarios where users have similar resources stored at multiple CSPs.

Our next step in data generation was to generate SWRL rules. These rules are generated using the meta-model described in Figure 6. We generated 200 SWRL rules based on the data generated in OWL files.

Performance of the Ontology Construction

In order to enable users to specify their access policy rules, the SBPMS first needs to provide them with information about all their resources. In this process, that is done only once at the beginning of the deployment of the framework, the SBPMS connects to knowledge bases of all the CSPs and retrieves information about all the resources the user has stored at the CSPs. This is essentially a construction of the global ontology in which the SBPMS gather information from OWL files of all the CSPs and construct the global ontology to be

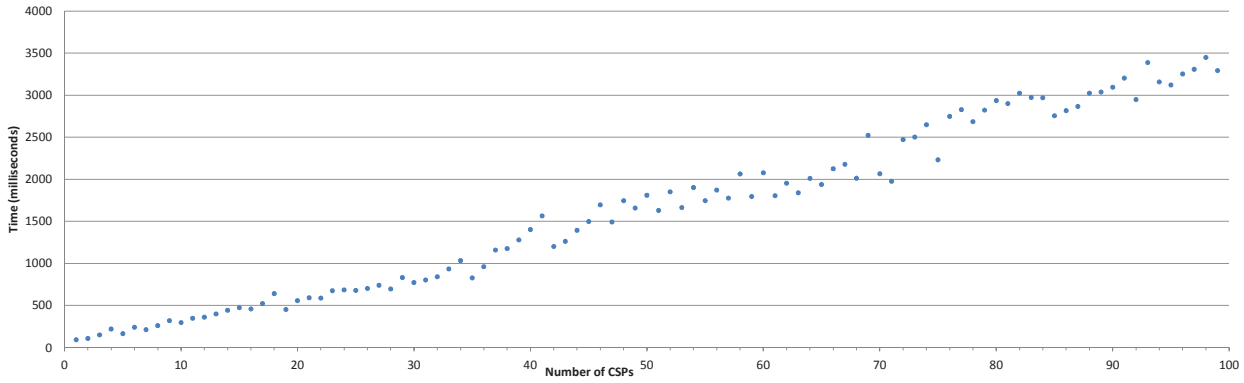


Figure 15: Global Ontology Construction Time

used for policy specification by the user. Figure 15 shows the performance of this process. As we can see, for the first CSP it takes less than 100 milliseconds to fetch required information to the server. For 10 CSPs, it takes about 300 milliseconds and for 40 CSPs the time it takes to fetch the ontologies is 1 second. For 100 CSPs, it takes less than 3.5 seconds to fetch all the ontologies stored at all the CSPs to the server; This may seem hight but we believe considering the large number of CSPs and size of the OWL files, it is an acceptable and reasonable performance. Also, note that this process is done only once at the beginning of the deployment of the framework.

Performance of the Authorization API

Next, we looked at performance of the authorization API and the provider authorization API. Whenever there is a change in one of the CSPs, for example, user adds a new object, adds a new subject, etc, we need to update the OWL file in the SBPMS side to reflect these changes. This experiment simulates the *pull* strategy that was discussed earlier. The results of this experiment are shown in Figure 16. As can be seen, when there is only one CSP it takes only 75 milliseconds to update the ontology base of the SBPMS server. If we increase the number of CSPs to 40, it takes 150 milliseconds to update the server. As demonstrated, if we increase the number of CSPs to 100, the time it takes to update the server when there is a change at the CSPs' side is less than 300 milliseconds in most cases.

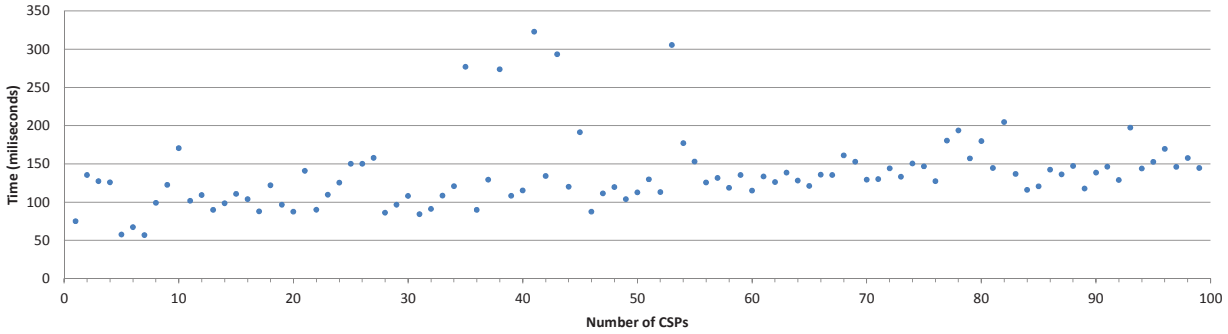


Figure 16: OWL Ontology Update Time

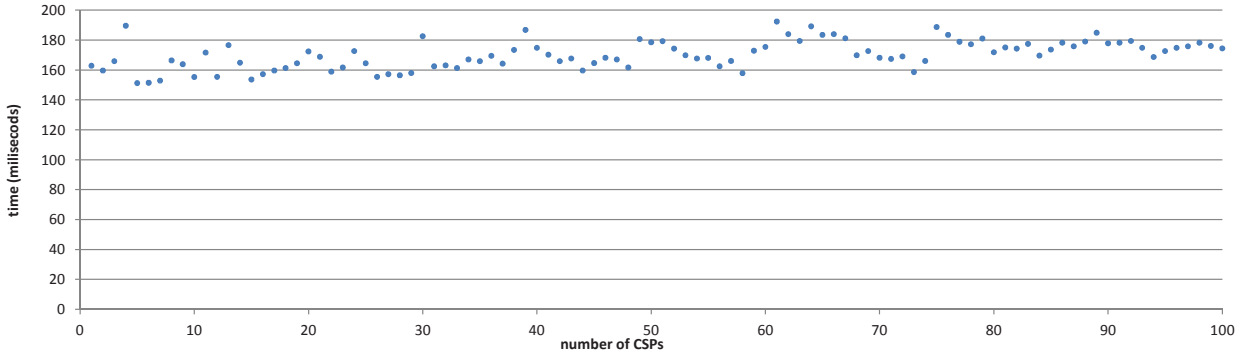


Figure 17: SWRL Rules Update Time

Similar to OWL files, when there is a change in SWRL rules at the SBPMS side, for example user adds a new rule, modifies an existing rule, deletes a rule, etc, these changes need to be reflected at the CSPs side too. The SBPMS should update SWRL rules files of the CSPs that are related to the change (determined by policy association module of the framework). This experiment simulates the *push* strategy that was discussed in section 4.3. The results of this experiment are shown in Figure 17. As it can be seen, for 100 CSPs it takes less than 200 milliseconds to update the policy base of the CSPs when there is a change in the policy base of the SBPMS server and the number of the CSPs does not have much effect on performance of this process.

Note that for all the experiments, we ran each of them five rounds and the results reported here are average of those five rounds of experiments. Overall, the results show that our proposed framework performs well and the time required for policy management is reasonable.

4.2 POLICY EVOLUTION COMPONENT

As it was discussed before, once RBAC is in place, maintenance of the system and dealing with policy evolution become important issues [89]. In this dissertation, we use the idea of role mining to deal with the policy evolution issues. Although there have been several attempts for role mining, there is no formal notion of goodness of a produced role set. Furthermore, the existing approaches do not consider the existing RBAC configuration and try to define everything from scratch, which is not acceptable for organizations that already have an RBAC system in place.

In this section, we formally define the problem of mining role hierarchy with minimal perturbation and present *StateMiner*, a heuristic solution to find an RBAC state as similar as possible to both the existing state and the optimal state. In order to achieve our goal, we use the theory of formal concept analysis [94], which has been shown to provide a strong theoretical foundation for role engineering [89]. We also introduce two different measures: structural complexity for optimality of an RBAC state, and similarity of sets of roles for

minimal perturbation. Our proposed algorithm, *StateMiner*, presents a heuristic solution to find an RBAC state as close as possible to both the deployed RBAC state and the optimal state. We then present evaluations and experimental results to demonstrate the effectiveness of our approach.

4.2.1 Overview: Formal Concept Analysis

In this section, we review the theory of formal concept analysis on which our work is based.

The input to formal concept analysis is called a formal context and defined as follows:

DEFINITION 1. *A formal context is a triple* (G, M, I) *where* G *and* M *are sets and* $I \subseteq G \times M$ *is a binary relation between* G *and* M*. The elements of* G *and* M *are called objects and attributes respectively. For* $g \in G$ *and* $m \in M$*, we write* gIm *when* $(g, m) \in I$*.*

In role mining, we see the user-permission relation as a formal context, where G is the set of all users, and M is the set of all permissions, and $(g, m) \in I$ if and only if the user corresponding to g has the permission corresponding to m.

DEFINITION 2. *A concept of the context* (G, M, I) *is a pair* (X, Y)*, where* $X \subseteq G$ *and* $Y \subseteq M$ *satisfy the following properties:*
$Y = \{m \in M | (\forall g \in X)\ gIm\}$*, i.e.,* Y *is the set of all attributes shared by all objects in* X
$X = \{g \in G | (\forall m \in Y)\ gIm\}$*, i.e.,* X *is the set of all objects that share all attributes in* Y*.*

X and Y are also called the *extent* and the *intent* of the concept (X, Y) respectively. The set of all concepts of the context is denoted by $\mathbf{B}(G, M, I)$. A concept (X_1, Y_1) is called a *subconcept* of (X_2, Y_2), shown as $(X_1, Y_1) \leq (X_2, Y_2)$ if and only if $X_1 \subseteq X_2$ (or, equivalently, $Y_2 \subseteq Y_1$). For example, Figure 18(b) shows the permissions associated with users as per Figure 1(a); here $(\{U_4, U_5\}, \{P_1, P_2, P_{11}, P_{12}\})$ is not a concept, because U_3, U_6 also have the permissions $\{P_1, P_2, P_{11}, P_{12}\}$. The pair $(\{U_3, U_4, U_5, U_6\}, \{P_1, P_2, P_{11},$
$P_{12}\})$ is a concept. The family of concepts complies the mathematical axioms defining a lattice, and is called a *concept lattice*. The concept lattice for the state in Figure 18(a) is shown in Figure 18(c). In this figure, the number to the right indicates the role, the first line shows the permissions assigned to the role, and the second line shows the users assigned to the role. In this concept lattice, each concept inherits all the permissions associated with

its subconcepts, and users are inherited in the other direction. Therefore, we can remove redundant permissions and users from each node. The result is called the *reduced concept lattice* and is given in Figure 18(d).

The reduced concept lattice defines a complete RBAC state. Each concept represents a role and the lattice can be viewed as the role hierarchy. In this RBAC state, each user is assigned to exactly one role, and each permission is assigned to exactly one role and the subconcept relation corresponds to the role inheritance relation. It is clear that the reduced concept lattice provides the semantic relationships among concepts and has more meanings than just a set of permissions. Using the reduced concept lattice as the role hierarchy has the disadvantage that the role hierarchy may be extremely large. In the reduced concept lattice, some concepts introduce no new users, no new permissions, or neither. However, it is not correct to remove all concepts with no new users or new permissions. We need to have a measure to compare the different role hierarchies generated from the reduced concept lattice and identify which one is more desirable which we will discuss in next section.

4.2.2 The Problem of Mining Role Hierarchy with Minimal Perturbation

In this section, we first define the role mining problem with minimal perturbation, then introduce two measures: one to measure goodness of an RBAC state and another to measure perturbation. Then, we formally define the problem of mining role hierarchy with minimal perturbation.

DEFINITION 3. *Given an RBAC state* $\gamma = \langle R, UA, PA, RH \rangle$*, find a new RBAC state that is consistent with access control configuration* $\rho = \langle U, P, UP \rangle$ *and is as close as possible to the existing RBAC state.*

The RBAC state is consistent with ρ if every user in U has the same set of authorized permissions in the RBAC state as in UP.

We adopt the example from [89] to use in this dissertation.

Example:. The original RBAC state is shown in Figure 18(a) which includes 10 users, 12 permissions, and 7 roles. Figure 18(b) shows the user-permission relation (UP) resulting from this state.

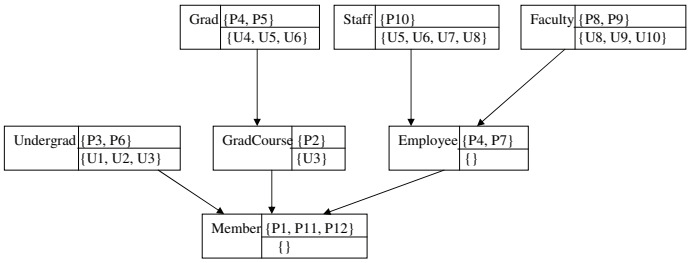
4.2.2.1 A Measure for Goodness of an RBAC State

An RBAC state is shown as $\gamma = \langle R, UA, PA, RH \rangle$, where R is a set of roles, $UA \subseteq U \times R$ is the user-role assignment relation, $PA \subseteq R \times P$ is the role-permission assignment relation, $RH \subseteq R \times R$ is a partial order over R, which is called a role hierarchy. Also an access control configuration is shown as $\rho = \langle U, P, UP \rangle$, where U is a set of users, P is a set of permissions, and UP is the user-permission assignment relation. Given an access control configuration, many RBAC states may be consistent with it. We need a to have a measurement of how good an RBAC state is in order to select among them.

Several metrics have been defined in the literature to measure goodness of identified roles. *Vaidya et al.* [81] suggest to minimize the number of roles. *Zhang et al.* [83] , *Ene et al.* [87], and *Lu et al.* [85] aim to minimize the number of user-role assignment and permission-role assignment relations. In [84] *Colantino et al.* describe a measure that minimize the administration cost of the resulting RBAC model. In [89] *Molloy et al.* propose the notion of weighted structural complexity that sums up the number of relationships in an RBAC state, with adjustable weights for different kinds of relationships. In [86] *Guo et al.* suggest to minimize the number of roles and the edges in role hierarchy graph. They argue that including UA and PA in weighted structural complexity is redundant because the role hierarchy incorporates the information represented by them. We believe that this argument does not hold as role hierarchy represents only relations between roles, while UA and PA are part of administration cost as well. The weighted structural complexity proposed in [89] allows direct user-permission assignments which we believe should not be allowed, because it is not clear when we should use roles and when we should use direct assignment of permissions to users; moreover, it defeats the purpose of RBAC.

Considering all the aforementioned metrics, the weighted structural complexity is the most general and most flexible measure that covers other measures as well. In order to have a measure, we adopt the notion of the *weighted structural complexity* as a measurement of goodness of an RBAC state, but unlike the approach by *Molloy et al.* in [89] we do not allow direct user permission assignments. The weighted structural complexity (WSC) is formally defined as follows:

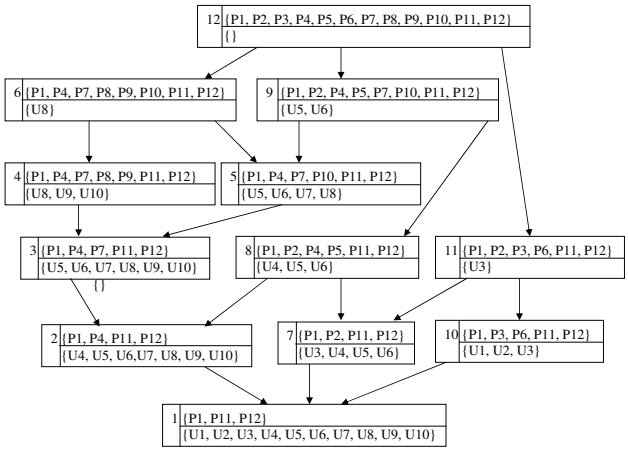
DEFINITION 4. *Given* $W = \langle w_r, w_u, w_p, w_h \rangle$, *where* $w_r, w_u, w_p, w_h \in Q^+ \cup \{\infty\}$,



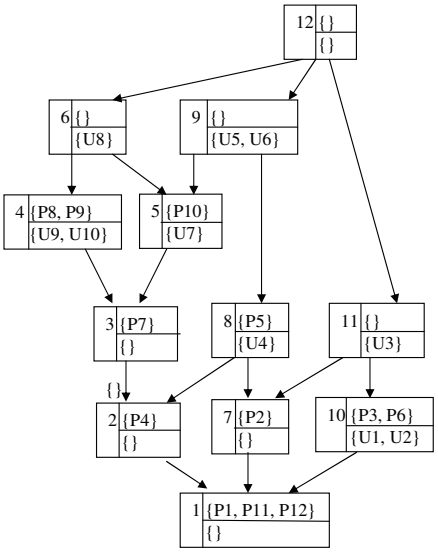
(a) The Original RBAC State

	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12
U1	1	0	1	0	0	1	0	0	0	0	1	1
U2	1	0	1	0	0	1	0	0	0	0	1	1
U3	1	1	1	0	0	1	0	0	0	0	1	1
U4	1	1	0	1	1	0	0	0	0	0	1	1
U5	1	1	0	1	1	0	1	0	0	1	1	1
U6	1	1	0	1	1	0	1	0	0	1	1	1
U7	1	0	0	1	0	0	1	0	0	1	1	1
U8	1	0	0	1	0	0	1	1	1	1	1	1
U9	1	0	0	1	0	0	1	1	1	0	1	1
U10	1	0	0	1	0	0	1	1	1	0	1	1

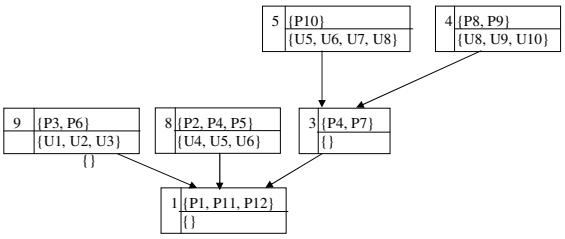
(b) The User-Permission Relation



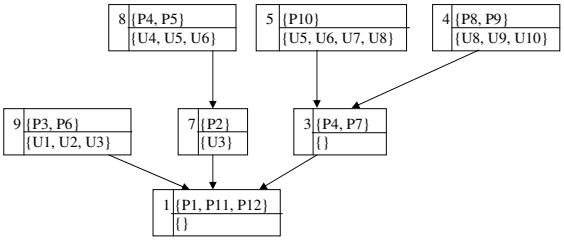
(c) The Concept Lattice



(d) The Reduced Concept Lattice



(e) The *StateMiner* Result for wf=0



(f) The *StateMiner* Result for wf=1

Figure 18: Example: The Original and Resulting RBAC States

the weighted structural complexity of an RBAC state γ, which is denoted as $wsc(\gamma, W)$, is computed as follows:

$wsc(\gamma, W) = w_r * |R| + w_u * |UA| + w_p * |PA| + w_h * |t\_r(RH)|$

where Q^+ is the set of all non-negative rational numbers, $|.|$ indicates the size of the set or relation, and $t\_r(RH)$ indicates the transitive reduction of role-hierarchy.

A transitive reduction is the minimal set of relationships that describes the same hierarchy. For example, $t\_r(\{(r_1, r_2),$
$(r_2, r_3), (r_1, r_3)\}) = \{(r_1, r_2), (r_2, r_3)\}$, as (r_1, r_3) can be inferred.

It is possible to adjust the weights of wsc to limit the RBAC states to meet different objectives. For example, by setting w_h to ∞, we can force a flat RBAC state since each role inheritance relation costs ∞ or by setting $w_r = 1, w_u = w_p = 0$, and $w_h = \infty$, we can minimize the number of roles.

4.2.2.2 A Measure for Minimal Perturbation Since we aim to mine role hierarchy with minimal perturbation, we define a metric to measure perturbation. For this, we need to find a way to measure similarity between the identified role set and the existing role set. In [88] *Vaidya et al.* use Jaccard Coefficient, a well known metric to compare the similarity of sample tests, to measure similarity between roles and role sets. Although Jaccard Coefficient is quiet straightforward, in general, it is too simple to measure similarity of sets of roles. Moreover, they only consider permissions directly assigned to roles while permissions acquired indirectly through role hierarchy relationships should also be taken into account. Even though role is basically a set of permissions, when it comes to computing similarity between roles, it is important to consider their positions in role hierarchies as well as the users who can acquire the roles.

We define a flexible and general measure for similarity between roles and role sets that takes into account users and permissions associated with roles as well as relations in role hierarchy with adjustable weights. First, we define a similarity measure between two roles and then extend it to two role sets. For similarity between two roles, we first define similarity based on permissions, users and relations in the role hierarchy individually, and then combine them to get a composite similarity measure.

Permission centric similarity between roles r_1 and r_2 is defined based on authorized permissions for roles as: $PermSim(r_1, r_2) = \frac{|rp_1 \cap rp_2|}{|rp_1 \cup rp_2|}$ where rp_1 is the set of all permissions authorized for role r_1 ($authorized\_permissions(r_1)$) and rp_2 is the set of all permissions authorized for role r_2 ($authorized\_permissions(r_2)$).

User centric similarity between roles r_1 and r_2 is defined based on authorized users for roles as: $UserSim(r_1, r_2) = \frac{|ru_1 \cap ru_2|}{|ru_1 \cup ru_2|}$ where ru_1 is the set of all users authorized for role r_1 ($authorized\_users(r_1)$) and ru_2 is the set of all users authorized for role r_2 ($authorized\_users(r_2)$).

One may say that two completely different roles maybe assigned to the same set of users leading to high $UserSim$ value which with a weight of 1/3 may outweigh the overall role similarity value. Although it is possible in theory, in reality if two completely different roles have same sets of $authorized\_users$, which we use in our approach instead of $assigned\_users$, they can be integrated to produce a more optimal RBAC state. Note that our goal here is to compare two sets of roles for the same organization and if two roles have same sets of $authorized\_users$ it is very likely that they are similar roles and should have high value of similarity.

Hierarchy Relation centric similarity between roles r_1 and r_2 is defined based on hierarchy relations for roles as: $RelSim(r_1, r_2) = \frac{min(|Sen(r_1)|,|Sen(r_2)|)}{max(|Sen(r_1)|,|Sen(r_2)|)} * \frac{1}{2}$ $+ \frac{min(|Jun(r_1)|,|Jun(r_2)|)}{max(|Jun(r_1)|,|Jun(r_2)|)} * \frac{1}{2}$ where $Sen(r)$ and $Jun(r)$ are the sets of immediate seniors and immediate juniors of role r respectively.

Since role hierarchies capture functional semantics in an organization, by including hierarchy relation centric similarity, we aim to retain this functional semantics in our measure. In measuring this similarity value, we assign same weights to senior and junior roles in a hierarchy. However, they could be adjusted depending on the needs of a system.

Now, by combining all these measures, we define similarity between two roles as a value between 0 and 1 as follows:

DEFINITION 5 (Role-Role Similarity). *For any two roles r_1 and r_2, we define similarity measure between them as: $sim(r_1, r_2) = PermSim(r_1, r_2) * w_{sp} + UserSim(r_1, r_2) * w_{su} + RelSim(r_1, r_2) * w_{sh}$ where $w_{sp} + w_{su} + w_{sh} = 1$.*

In our experiments, we set the weights as $w_{sp} = w_{su} = w_{sh} = \frac{1}{3}$. However, they could be adjusted based on the system and application requirements. When two roles are identical

with regards to all permissions, users and relations, their similarity is 1 (such as role "Employee" in Figure 18(a) and role "3" in Figure 18(d)), and when two roles have mutually exclusive permission sets, mutually exclusive user sets and have no similar relations in the hierarchy, their similarity is 0.

Measuring the similarity between two sets of roles is a significantly more complex task than measuring the similarity between two roles. It is not clear whether a single role corresponds to only one other role or to a set of roles. Similarly, it is unclear if a role can be matched to more than one roles in the other role set. We extend Role-Role Similarity measure to measure similarity between two sets of roles; we compute similarity value for each pair of roles in both role sets, sort them and then pick the maximum similarity value of pairs of roles and take their average. A key issue is deciding how many of these similarity values should be taken into account specially when two role sets have different numbers of roles. Two intuitive approaches would be as follows:

(1) take smaller role set and compute the average over maximum similarity values for each role in this set. However, this is not a good measure when we have the larger second set and maximum similarity value for each role of the first set turns out to be 1 (i.e., each role in the first set has an exactly matching role in the second set). The two sets have similarity value of 1 indicates that they are exactly similar - which is not the case as the set sizes are different (VAG algorithm, the approach proposed by *Vaidya et al.* in [88], falls into this category as it considers average over the size of the mined set, which could be the smaller set).

(2) take average over similarity values of all pairs of roles from the two sets. This can be bad in several situations. For instance, consider a scenario with two exactly similar role sets with identical roles which are disjoint from each other. In such a case, for each role in the first role set there is exactly one role in the second role set for which similarity value is 1 and for all other roles similarity value is 0. Hence, if we take the average over all pairs, the final similarity value here would be $\frac{1}{|rs|}$ ($|rs|$ is size of the role sets), which is not an intuitive value to indicate exactly matching sets.

Algorithm 1 Similarity: $\mathrm{Sim}(rs_1, rs_2)$

Input: two role sets rs_1, rs_2 where rs_1 is the smaller role set

Input: weights for permissions, users and relations $\langle w_{sp}, w_{su}, w_{sh} \rangle$

Input: t as threshold

1: $Counter \leftarrow 0$
2: $SimMatrix \leftarrow 0$
3: $Avgsim \leftarrow 0$
4: $Sim \leftarrow 0$
5: $MaxSim \leftarrow 0$
6: **for** each role $r_1 \in rs_1$ **do**
7: $\quad rp_1 \leftarrow authorized\_permissions(r_1)$
8: $\quad ru_1 \leftarrow authorized\_users(r_1)$
9: $\quad$ **for** each role $r_2 \in rs_2$ **do**
10: $\quad\quad rp_2 \leftarrow authorized\_permissions(r_2)$
11: $\quad\quad ru_2 \leftarrow authorized\_users(r_2)$
12: $\quad\quad np_i \leftarrow |rp_1 \cap rp_2|$
13: $\quad\quad nu_i \leftarrow |ru_1 \cap ru_2|$
14: $\quad\quad np_u \leftarrow |rp_1 \cup rp_2|$
15: $\quad\quad nu_u \leftarrow |ru_1 \cup ru_2|$
16: $\quad\quad PermSim \leftarrow \frac{np_i}{np_u}$
17: $\quad\quad UserSim \leftarrow \frac{nu_i}{nu_u}$
18: $\quad\quad RelSim \leftarrow \frac{min(|Sen(r_1)|,|Sen(r_2)|)}{max(|Sen(r_1)|,|Sen(r_2)|)} * \frac{1}{2} + \frac{min(|Jun(r_1)|,|Jun(r_2)|)}{max(|Jun(r_1)|,|Jun(r_2)|)} * \frac{1}{2}$
19: $\quad\quad Sim \leftarrow PermSim * w_{sp} + UserSim * w_{su} + RelSim * w_{sh}$
20: $\quad\quad SimMatrix[r_1][r_2] \leftarrow Sim$
21: $\quad\quad MaxSim \leftarrow Max(Sim, MaxSim)$
22: $\quad\quad Sim \leftarrow 0$
23: $\quad$ **end for**
24: $\quad Avgsim \leftarrow Avgsim + MaxSim$
25: $\quad Counter \leftarrow Counter + 1$
26: $\quad MaxSim \leftarrow 0$
27: **end for**

Algorithm 1 Similarity: Sim(rs_1, rs_2) (continued)

```
for each role r_2 ∈ rs_2 that has not been matched do
    MaxSim ← Max_{r_1 ∈ rs_1} SimMatrix[r_1][r_2]
    if MaxSim ≥ t then
        Avgsim ← Avgsim + MaxSim
        Counter ← Counter + 1
    end if
end for
Avgsim ← Avgsim / Counter
Return Avgsim
```

Ideally, we want every role to contribute to the final similarity measure. Wa propose the following approach to measure the similarity between two role sets rs_1 and rs_2. Without loss of generality, we assume that rs_1 is the smaller role set.

DEFINITION 6 (Role Set-Role Set Similarity). *For any two role sets rs_1 and rs_2 where rs_1 is the smaller role set, we compute similarity between them as follows:*

Step 1: $\forall r_i \in rs_1$, find $Max_{r_j \in rs_2} sim(r_i, r_j)$ such that for all selected pairs (r_i, r_j) and (r_x, r_y), if $r_i \neq r_x$ then $r_j \neq r_y$. In this step every role in rs_1 is matched with exactly one distinct role in rs_2, but there are some roles in rs_2 that have not been matched with any role from rs_1.

Step 2: For those roles in rs_2 that have not been matched in the first step, we consider only the roles that have a similarity value above a predefined threshold.

Step 3: Finally, take the average over all of the chosen similarity values.

In Step 2, two naive approaches could be to include all the roles for computing the similarity values or ignore all of them; each of which has disadvantages. If we ignore all of them there maybe some roles with high similarity values that do not contribute to the final similarity value. Similarly, if we include all of them there maybe some roles with very low similarity values that lead to a low overall similarity value. Therefore, we use a threshold based approach. In our experiments we use $t = 0.5$ as threshold and in Step 2 consider only the roles that have a similarity value above 0.5.

We also define the dissimilarity between two role sets as follows: $dissim(rs_1, rs_2) =$

$1 - sim(rs_1, rs_2)$. The algorithm for computing similarity is shown in Algorithm 1.

4.2.2.3 Global Optimization Function

In general, the problem of mining role hierarchy with minimal perturbation can be seen as a multi objective optimization problem that aims to trade-off conflicting objectives [95]. Usually there are a number of objectives that should be minimized or maximized. Here we have two objectives: (1) minimize the weighted structural complexity of the resulting RBAC state, and (2) maximize the similarity (minimize the dissimilarity) between identified roles and the existing roles. The simplest way to define a global optimum is to compute a weighted sum of all the objective functions. The weighted similarity is a number between 0 and 1 but the wsc is not, so we multiply sim by wsc to bring both numbers to a comparable range. We define the following global optimization function:

Definition 7 (Global Optimization Function). *Given a weighted structural complexity, wsc, of an RBAC state, and a dissimilarity measure, dissim, between two sets of roles, global optimization function is defined as:*

$GOF(wsc, dissim) = (1 - wf) * wsc + wf * wsc * dissim$ *where* $wf \in [0, 1]$ *is a user defined weighting factor for the similarity.*

As we mentioned before, $dissim(rs_1, rs_2) = 1 - sim(rs_1, rs_2)$, so by replacing $dissim$ with sim the global optimization function can be computed by $GOF(wsc, sim) = wsc * (1 - wf * sim)$.

As we can see, if the weight factor for similarity is zero ($wf = 0$), we have $GOF = wsc$ which means we do not care about how similar the result is to the existing configuration and the only important measure here is weighted structural complexity. Also if the weight factor for similarity is one ($wf = 1$), we have $GOF = wsc * dissim$ which means we do not care about structural complexity of the result; the only important measure in this case is $dissim$. In the later case as we multiplied $dissim$ by wsc to bring both measure to the same range, we can delete wsc, so we have $GOF = dissim = 1 - sim$. We now define the problem of mining role hierarchy with minimal perturbation as follows:

Definition 8 (The problem of Mining Role Hierarchy with Minimal Perturbation). *Given a deployed RBAC state* $\gamma_{dpl} = \langle R_{dpl}, UA_{dpl}, PA_{dpl}, RH_{dpl} \rangle$ *and a system*

configuration $\rho = \langle U, P, UP \rangle$*, find an RBAC state* $\gamma_{new} = \langle R_{new}, UA_{new}, PA_{new}, RH_{new} \rangle$ *consistent with* UP *such that it minimizes* $GOF(wsc(\gamma_{new}, W), dissim(R_{dpl}, R_{new})) = wsc(\gamma_{new}, W) * (1 - wf * sim(R_{dpl}, R_{new}))$.

The goal of mining role hierarchy with minimal perturbation is to minimize the global optimization function of the predefined optimality measure (i.e., weighted structural complexity of γ_{new}) and the predefined perturbation measure (i.e., dissimilarity between R_{dpl} and R_{new}).

4.2.3 The StateMiner Algorithm

In this section, we present a heuristic algorithm to find a set of roles satisfying the goals of mining role hierarchy with minimal perturbation. The algorithm consists of two phases. In the first phase, we generate the reduced concept lattice using the deployed configuration, which gives us an RBAC state. In the second phase, we prune this lattice and select the final RBAC state. In order to do this, we use a greedy algorithm to heuristically optimize the lattice.

Once we have the reduced concept lattice, we should decide which roles are appropriate and which ones should be removed. Removing each role reduces the cost of creating the role and the associated relationships. However, we need to add back some relationships so that user-permission assignment relation and the inheritance relation remain correct. We use one general pruning rule: we remove role r from the reduced concept lattice when the value of the GOF decreases after removing that role.

StateMiner as shown in Algorithm 2 is a greedy algorithm; it iterates over all of the roles in the reduced concept lattice and performs pruning if the change will decrease global optimization function of wsc of the RBAC state and the similarity between the deployed role set and the role sets with and without that role. It stops when no more operations can be performed. Since if a role r in the reduced concept lattice has no new permissions and no new users, it is more likely to be removed, we first check this kind of roles. Next, we check roles with no new permissions, roles with no new users, and finally roles with both new permissions and new users. Unlike *StateMiner*, *HierarchicalMiner* [89] does not consider the

Algorithm 2 Minimal Perturbation Problem in Presence of Role Hierarchy: $StateMiner(\gamma, W, wf)$

Input: current RBAC state $\gamma_{dpl} = \langle R_{dpl}, UA_{dpl}, PA_{dpl}, RH_{dpl} \rangle$

Input: weight factors for complexity, $W\langle w_r, w_u, w_p, w_h \rangle$

Input: system configuration $\rho = \langle U, P, UP \rangle$

Input: weight factor for similarity, $wf \in [0, 1]$

1: create reduced concept lattice $\gamma_{rcl} = \langle R_{rcl}, UA_{rcl}, PA_{rcl}, RH_{rcl} \rangle$

2: Sort R_{rcl} such that order is first roles with neither new

3: users nor new permissions, roles with no new permissions,

4: permissions, roles with no new users, and finally

5: roles with both new users and new permissions

6: $wsc_{before} \leftarrow wsc(\gamma_{rcl}, W)$

7: $sim_{before} \leftarrow sim(R_{dpl}, R_{rcl})$

8: $\gamma_{new} \leftarrow \gamma_{rcl}$

9: **for** each role $r \in R_{rcl}$ **do**

10: $Sen(r) \leftarrow \{r_i \in R_{rcl} | (r_i, r) \in t\_r(RH_{rcl})\}$

11: $Jun(r) \leftarrow \{r_j \in R_{rcl} | (r, r_j) \in t\_r(RH_{rcl})\}$

12: $rp \leftarrow assigned\_permissions(r)$

13: $ru \leftarrow assigned\_users(r)$

14: $\forall u \in ru\ \forall r_i \in Jun(r)\ UA_{new} \leftarrow UA_{new} \cup (u, r_i)$

15: $\forall u \in ru\ UA_{new} \leftarrow UA_{new} \setminus \{(u, r)\}$

16: $\forall p \in rp\ \forall r_j \in Sen(r)\ PA_{new} \leftarrow PA_{new} \cup (p, r_j)$

17: $\forall p \in rp\ PA_{new} \leftarrow PA_{new} \setminus \{(p, r)\}$

18: $\forall r_i \in Sen(r)\ \forall r_j \in Jun(r)\ RH_{new} \leftarrow RH_{new} \cup (r_i, r_j)$

19: $\forall r_i \in Sen(r)\ RH_{new} \leftarrow RH_{new} \setminus \{(r_i, r)\}$

20: $\forall r_j \in Jun(r)\ RH_{new} \leftarrow RH_{new} \setminus \{(r, r_j)\}$

21: $R_{new} \leftarrow R_{new} \setminus \{r\}$

22: $Compute\ t\_r(RH_{new})$

23: $wsc_{after} \leftarrow wsc(\gamma_{new}, W)$

24: $sim_{after} \leftarrow sim(R_{dpl}, R_{new})$

Algorithm 2 Minimal Perturbation Problem in Presence of Role Hierarchy: $StateMiner(\gamma, W, wf)$ (continued)

```
    if wsc_after * (1 − wf * sim_after) < wsc_before * (1 − wf * sim_before) then
        γ_rcl ← γ_new
        wsc_before ← wsc_after
        sim_before ← sim_after
    end if
end for
Return γ_rcl
```

concepts with both new users and new permissions in the pruning process.

If we run the algorithm using the original RBAC state in Figure 18(a), the configuration in Figure 18(b), $w_r = w_u = w_p = w_h = 1$, the output is shown in Figures 18(e) and 18(f) for and $wf = 0$ and $wf = 1$ respectively. As we mentioned before, when $wf = 0$ the only important measure is weighted structural complexity. The original RBAC state (Figure 18(a)) has $wsc = 40$, the one found by *HierchicalMiner* [89] has $wsc = 40$, while the one found by *StateMiner* (Figure 18(e)) has $wsc = 37$. Moreover, as shown in Figure 18(f) when $wf = 1$ the mined RBAC state is exactly same as the original one and has $wsc = 40$.

Time Complexity

As the *StateMiner* algorithm has two phases, its computational complexity depends on the complexity of the generation of the reduced concept lattice and the pruning process. The time complexity for generating the reduced concept lattice is linear in the size of the lattice. Since every concept in the reduced concept lattice is a role, the size of the lattice is the number of roles. The number of roles in the worst case equals the number of permissions, n, when each permission is a role by its own. As the algorithm iterates n times during pruning process, the worst case cost is $O(n^2)$ where n is the number of permissions in the system.

4.2.4 Evaluation and Experimental Results

We have implemented the proposed algorithm. For generating concepts we use Colibri-Java by Daniel Gotzmann [97] which is based on [96]. The heuristic approach for pruning is

written in Java too and implements parsing, data structures, and the algorithms.

In this section, we evaluate the effectiveness of the *StateMiner* algorithm. The dataset we use is a synthetic dataset used to evaluate the *HiararchicalMiner* by *Molloy et al.* [89]. The data set contains 493 users and 56 permissions. As we mentioned earlier, the goal of the *StateMiner* algorithm is to find an RBAC state that has the smallest weighted structural complexity and is as similar as possible to the existing RBAC state. By tuning the weight factor, organizations can set their priorities on the relative importance of maintaining the resulting state against the cost of changing the current state. It means that an organization can tune the trade-off between the maintenance cost and the change over cost. Furthermore, by assigning weight factors for complexity they can set priorities on the importance of each kind of relationship in the system.

Table 2 shows the weighted structural complexities of the original RBAC state and the states generated by *StateMiner* for different similarity weight factors. We also include the result for *HierarchicalMiner* and Optimal search algorithm that aims to find optimal RBAC state [89]. As we discussed in section 4.2.2.3 if the weight factor for similarity is zero (wf = 0), the only important measure here is weighted structural complexity and *StateMiner* covers *HierarchicalMiner* algorithm. We can see that *StateMiner* with $wf = 0$ has smaller wsc than the HierarchicalMiner and is closer to the optimal solution. The reason is that in the pruning process, *StateMiner* considers roles with both new users and new permissions while *HierarchicalMiner* does not. Note that *StateMiner* with $wf \geq 0.4$ has larger wsc than *HierarchicalMiner*. This is to be expected as *HierarchicalMiner* does not consider existing RBAC state while *StateMiner* does and when we aim to to keep the identified role set closer to the original state the cost will be more with regards to wsc. For the same reason, as wf increases, wsc increases as well.

Since the VAG algorithm [88] is the only algorithm in the literature that considers existing roles in mining process, we have compared our proposed similarity measure with theirs. The VAG algorithm only supports flat RBAC and considers only permissions directly assigned to roles while in our proposed similarity measure permissions acquired indirectly through role hierarchy relationships are also considered as well as positions of roles in role hierarchy and users who can acquire the roles. In order to compare the two measures, we implemented

Table 2: The *StateMiner* Algorithm Experimental Results

	$W = \{1,1,1,1\}$					$W = \{1,1,2,2\}$				
	R	UA	PA	RH	WSC	R	UA	PA	RH	WSC
Original [a]	32	799	35	19	885	32	799	35	19	939
wf=0	22	493	67	20	602	21	498	67	18	689
wf=0.1	21	498	67	18	604	21	498	67	18	689
wf=0.2	22	498	65	20	605	22	498	65	20	690
wf=0.3	21	498	65	18	605	22	498	65	20	690
wf=0.4	21	498	70	18	607	22	498	65	20	690
wf=0.5	21	498	70	19	608	22	498	68	20	696
wf=0.6	21	498	70	19	608	22	498	68	20	696
wf=0.7	23	498	65	25	611	22	498	68	21	698
wf=0.8	23	498	65	25	611	23	498	66	23	699
wf=0.9	24	498	63	27	612	24	498	63	27	702
wf=1	24	498	63	27	612	24	498	63	27	702
HierarchicalMiner	21	498	67	19	605	21	505	67	20	696
optimal	19	496	59	14	600	19	496	57	16	685

[a] Numbers of roles (R) and user-permission assignment relations (PA) in original state are different from the ones reported in table 2 in [89]. We believe our numbers are correct because if we calculate total cost using reported numbers in [89], it is 905 instead of 875 ($22 + 799 + 65 + 19 + 0 = 905$) in case of $W = \{1,1,1,1,1\}$ and 989 instead of 959 ($22 + 799 + 2 * 65 + 2 * 19 + 2 * 0 = 989$) in case of $W = \{1,1,2,2,2\}$.

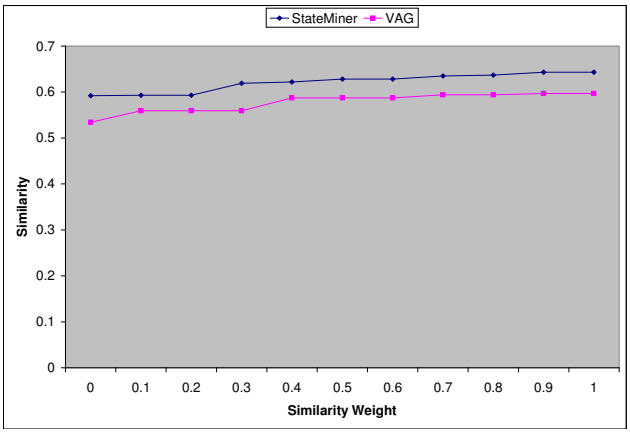


Figure 19: Comparison with VAG

the VAG algorithm and applied it to the same data set as our proposed algorithm. The goal is to compare the resulting mined role sets of the two approaches with the existing role set and determine which one generates a role set more similar to the existing role set. Figure 19 shows similarity of the mined RBAC state using our proposed similarity measure and using VAG algorithm. We observe that compared to the VAG algorithm, our approach provides better results and the mined roles of our proposed similarity measure are closer to the original state.

Figure 21 depicts the roles related to students portion of the original and generated states; Figure 21(a) shows the original RBAC state while figure 21(b) and 21(c) show the mined results of *StateMiner* for $wf = 0$ and $W = \{1, 1, 1, 1\}$ and $W = \{1, 1, 2, 2\}$ respectively. We can see that *StateMiner* produces semantically meaningful roles. Figure 21(b) represents the same result produced by *HierarchicalMiner*. This is because $wf = 0$.

We have also run some experiments to check the effect of weight factor on the results. Figure 20 shows GOF as a function of the similarity weight factor. As expected, by increasing wf, similarity and wsc increases while GOF decreases.

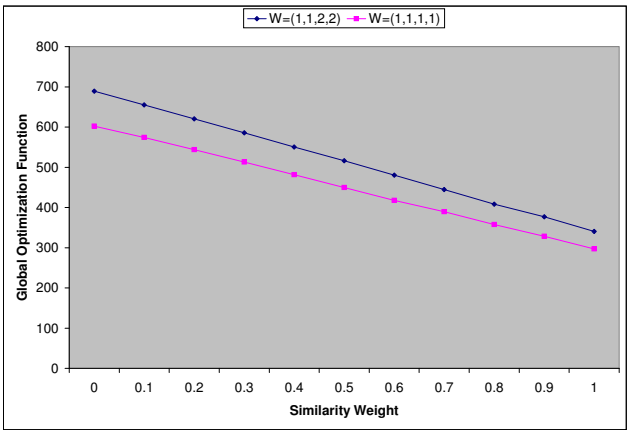
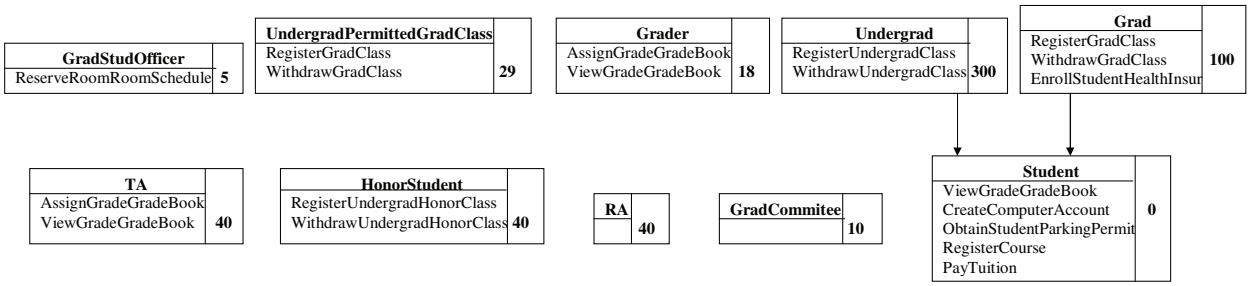
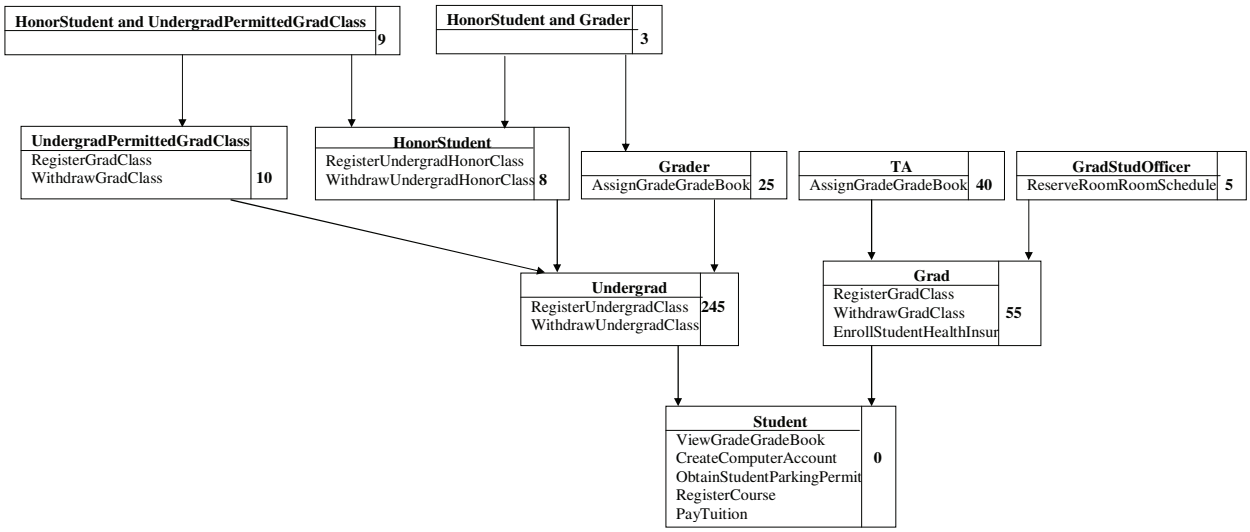


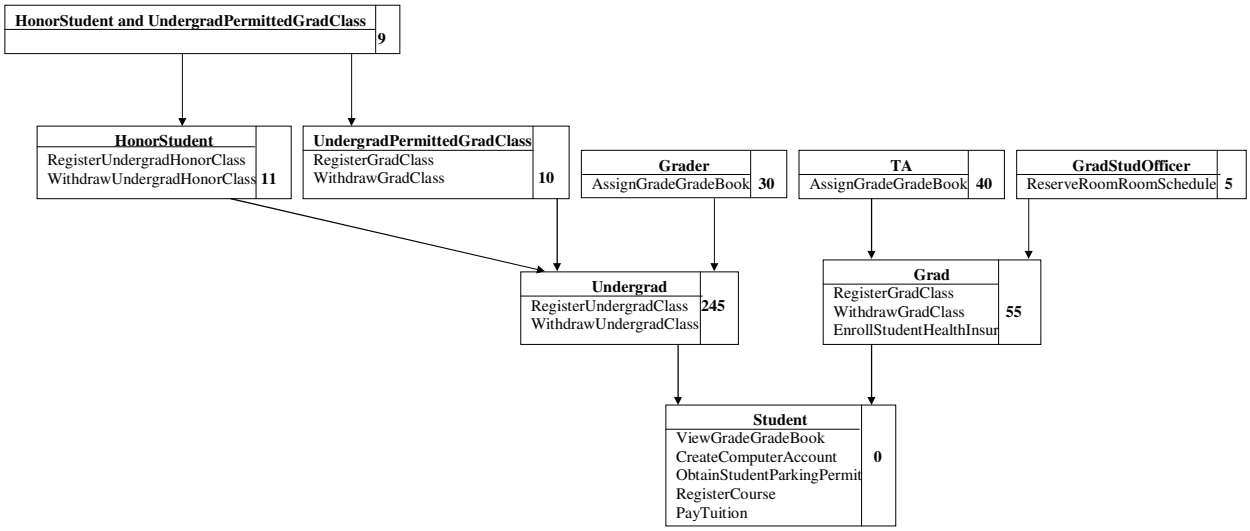
Figure 20: Global Optimization Function vs. Weight



(a) The Original State



(b) The *StateMiner* Result for W={1,1,1,1}



(c) The *StateMiner* Result for W={1,1,2,2}

Figure 21: Mining Results for the University Dataset

5.0 CONCLUSIONS AND FUTURE WORK

In this dissertation, we proposed research tasks to address challenges in policy management and access control in cloud computing environments. In particular, we addressed challenges related to unified policy management and policy specification, and policy evolution.

With regards to Task I, we proposed Policy Management as a Service (PMaaS), a cloud based policy management framework that puts users in full control of their resources which may be scattered across multiple CSPs. It is designed to give cloud users a unified control point for specifying authorization policies, who and what can get access to their data, content, and services, no matter where all those things live on the Cloud. It relies on a user's centrally located policy manager of those resources and enables users to manage access policies using a centralized policy manager which provides capabilities for specifying access policies and exporting them to the CSPs on behalf of the user.

Then, we presented lessons we learned from a case study where we implemented a unified policy management system for various real world CSPs. Based on those lessons and motivated by limitations of existing approaches, we proposed a semantic based policy management framework that is designed to help CSPs to define and manage security policies using semantic web technologies and allows interoperation among different CSPs. We presented our proposed framework and described its components. We also introduced semantic based policy specification to provide a common understandable semantic basis for policy specification in cloud computing environments. Moreover, we defined OWL ontologies that represent temporal constraints in GTRBAC and show how they can be used to specify and model temporal constraints. Furthermore, we presented a proof of concept implementation of the proposed framework to show its applicability and reported results of the experiments we performed to evaluate performance of the framework.

With regards to Task II, we discussed how to use role mining techniques for policy evolution purpose. We formally defined the problem of mining a role hierarchy with minimal perturbation. We also introduced two measures: a measure for goodness of an RBAC state and a measure for minimal perturbation, then based on these measures we developed StateMiner, a heuristic solution to find an RBAC state as close as possible to both the deployed RBAC state and the optimal state. We also performed some experiments to demonstrate the effectiveness of the proposed algorithm.

There are several future work related to the research presented in this dissertation. First, We can extend the PMaaS by offering policy recommendation capabilities where the framework can analyze the policies defined by user and use machine learning techniques for example to predict policies and recommend them to user. Of course, user will have the option to reject or modify the recommended policies.

Second, our approach in policy evolution component does not consider separation of duty constraints and it is not clear what effects this migration will have on the existing separation of duty constraints. It could be extended to consider separation of duty constrains and their effects in the process of migrating to RBAC.

BIBLIOGRAPHY

[1] P. Mell and T. Grance, "The NIST Definition of Cloud Computing," NIST Special Publication 800-145, Available from http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf, 2011.

[2] Cloud Security Alliance, "Security Guidance for Critical Areas of Focus in Cloud Computing V3.0", Available from https://cloudsecurityalliance.org/guidance/csaguide.v3.0.pdf, 2011.

[3] D. Catteddu and G. Hogben, "Cloud Computing: Benefits, risks and recommendations for information security," ENISA 2009; http://www.enisa.europa.eu/act/rm/files/deliverables/cloud-computing-risk-assessment/at_download/fullReport.

[4] P.J. Bruening and B.C. Treacy, "Cloud Computing: Privacy, Security Challenges," Bureau of National Affairs, Inc., 2009; www.hunton.com/files/tbl_s47Details/FileUpload265/2488/CloudComputing_Bruening-Treacy.pdf.

[5] Y. Chen, V. Paxson, and R. H. Katz, "Whats New About Cloud Computing Security?" , Technical Report No. UCB/EECS-2010-5, EECS Dept., University of California at Berkeley, 2010; www.eecs.berkeley.edu/Pubs/TechRpts/2010/EECS-2010-5.html.

[6] T. Jaeger and J. Schiffman, "Outlook: Cloudy with a Chance of Security Challenges and Improvements", IEEE Security and Privacy, Vol. 8, No. 1, pp. 77-80, 2010.

[7] N. Gruschka and M. Jensen, "Attack Surfaces: A Taxonomy for Attacks on Cloud Services", In Proc. of the 3rd IEEE International Conference on Cloud Computing (Cloud 2010), pp. 276-279, Miami, FL, USA, 2010.

[8] X. Tian, X. Wang, and A. Zhou, "DSP RE-Encryption: A Flexible Mechanism for Access Control Enforcement Management in DaaS", In Proc. of the 2nd IEEE International Conference on Cloud Computing (Cloud 2009), pp. 25-32, Bangalore, India, 2009.

[9] M. Jensen, J. Schwenk, N. Gruschka, and L. L. Iacono, "On Technical Security Issues in Cloud Computing", In Proc. of the 2nd IEEE International Conference on Cloud Computing (Cloud 2009), pp. 109-116, Bangalore, India, 2009.

[10] B. R. Kandukuri, R. Paturi , and A. Rakshit, "Cloud Security Issues", In Proc. of the 6th IEEE International Conference on Services Computing (SCC'09), pp. 517-520, Bangalore, India, 2009.

[11] J. B.D. Joshi, E. Bertino, U. Latif, and A. Ghafoor, "A Generalized Temporal Role-Based Access Control Model," IEEE Transactions on Knowledge and Data Engineering, Vol. 17, No. 1, 2005.

[12] M. Kim, J. B.D. Joshi, M.K. Kim, "Access Control for Cooperation Systems based on Group Situation," In Proc. of the 4th International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom2008), USA, 2008.

[13] S. Chakraborty and I. Ray, "TrustBAC: integrating trust relationships into the RBAC model for access control in open systems," In Proc. of the 11th ACM symposium on access control models and technologies (SACMAT06), pages 49-58, ACM Press, 2006.

[14] N. B. Kodali, C. Farkas, and D. Wijesekera, "Specifying Multimedia Access Control using RDF," Journal of Computer Systems, Science and Engineering, Vol. 19, 2004.

[15] R. Bhatti, J. B. D. Joshi, E. Bertino, and A. Ghafoor, "Access Control in Dynamic XML-based Web-Services with X-RBAC," The First International Conference in Web Services, USA, 2003.

[16] R. Bhatti, J. B. D. Joshi, E. Bertino, and A. Ghafoor, "X-GTRBAC An XML-based Policy Specification Framework and Architecture for Enterprise-Wide Access Control," ACM Transactions on Information and System Security Vol. 8, No. 2, 2005.

[17] S. M. Chandran and J. B.D. Joshi, "LoT RBAC: A Location and Time-based RBAC Model", In Proc. of the 6th International Conference on Web Information Systems Engineering (WISE 2005), New York, Nov 2005.

[18] E. Bertino, F. Paci, R. Ferrini, "Privacy-preserving Digital Identity Management for Cloud Computing," IEEE Computer Society Data Engineering Bulletin, pages 1-4, 2009.

[19] M. Ko, G.J. Ahn, and M. Shehab "Privacy enhanced User-Centric Identity Management," In Proceedings of IEEE International Conference on Communications, Dresden, Germany, June 14-18, 2009.

[20] J. B.D. Joshi, R. Bhatti, E. Bertino, and A. Ghafoor, "Access-Control Language for Multidomain Environments," IEEE Internet Computing, Vol. 8, No. 6, 2004.

[21] Y. Zhang and J. B.D. Joshi, "Access Control and Trust Management for Emerging Multidomain Environments," in Annals of Emerging Research in Information Assurance, Security and Privacy Services, Editors: S. Upadhyaya, R. O. Rao 2009.

[22] M. Blaze, S. Kannan, I. Lee, O. Sokolsky, J. M. Smith, A. D. Keromytis, and W. Lee, "Dynamic Trust Management," IEEE Computer, pages 44-51, 2009.

[23] D. Shin and G.J. Ahn, "Role-based Privilege and Trust Management," Computer Systems Science and Engineering Journal, Vol. 20, No. 6, CRL Publishing, 2005.

[24] Gail-Joon Ahn, Hongxin Hu and Jing Jin, "Security-enhanced OSGi Service Environments," IEEE Transactions on Systems, Man, and Cybernetics-Part C: Applications and Reviews, Vol. 39, No. 5, 2009.

[25] L. Teo and G.J. Ahn, "Managing Heterogeneous Network Environments Using an Extensible Policy Framework," In Proc. of the ASIAN ACM Symposium on Information, Computer and Communications Security (ASIACCS07), Singapore, ACM Press, 2007.

[26] H. Takabi, M. Kim, J. B.D. Joshi, and M. B. Spring, "An architecture for specification and enforcement of temporal access control constraints using OWL," In Proc. of the 2009 ACM workshop on Secure web services (SWS'09), ACM Press, 2009.

[27] H. Takabi and J. B.D. Joshi, "StateMiner: An Efficient Similarity-Based Approach for Optimal Mining of Role Hierarchy," In Proc. of the 15th ACM symposium on access control models and technologies (SACMAT10), USA, ACM Press, 2010.

[28] H. Takabi, J. B.D. Joshi, and G.J. Ahn, "Security and Privacy Challenges in Cloud Computing Environments," IEEE Security and Privacy, Vol. 8, No. 6, pp. 25-31, 2010.

[29] H. Takabi, J. B. D. Joshi, and G.J. Ahn, "SecureCloud: Towards a Comprehensive Security Framework for Cloud Computing Environments," In Proc. of the 1st IEEE International Workshop Emerging Applications for Cloud Computing (CloudApp 2010), pp. 393-398, Seoul, South Korea, 2010.

[30] H. Takabi and J. B.D. Joshi, "Policy Management as a Service: An Approach to Manage Policy Heterogeneity in Cloud Computing Environment," In Proc. of the 45th Hawaii International Conference on System Sciences (HICSS), pp. 5500-5508, Hawaii, USA, January 4-7, 2012.

[31] H. Takabi and J. B. D. Joshi, "Semantic Based Policy Management for Cloud Computing Environments," International Journal of Cloud Computing, Vol. 1, No. 2, 2012.

[32] H. Takabi and J. B.D. Joshi, "Toward a Semantic Based Policy Management Framework for Interoperable Cloud Environment," In Proc. of the International IBM Cloud Academy Conference (ICA CON 2012), Research Triangle Park (RTP), North Carolina, USA, April 19-20, 2012.

[33] M. I. Yague, A. Mana, J. Lopez, and J. M. Troya, "Applying the Semantic Web Layers to Access Control", In Proc. of the 14th International Workshop on Database and Expert Systems Applications (DEXA'03), pp. 622-626 , 2003.

[34] J. M. Marn Prez, J. Bernal Bernab, J. M. Alcaraz Calero, F. J. Garcia Clemente, G. Martnez Prez, and A. F. Gmez Skarmeta, "Semantic-based authorization architecture

for Grid", Journal of Future Generation Computer Systems archive, Vol. 27, No. 1, pp. 40-55, 2011.

[35] L. Hu, Sh. Ying, X. Jia, and K. Zhao, "Towards an Approach of Semantic Access Control for Cloud Computing", In Proc. of the 1st International Conference on Cloud Computing (CloudCom'09), pp. 145-156, Beijing, China, 2009.

[36] J. M. Alcaraz Calero, N. Edwards, J. Kirschnick, L. Wilcock, and M. Wray, "Toward a Multi-Tenancy Authorization System for Cloud Services", IEEE Security and Privacy, Vol. 8, No. 6, pp. 48-55, 2010.

[37] P. Mitra, C.C. Pan, P. Liu, and V. Atluri, "Privacy-preserving semantic interoperation and access control of heterogeneous databases". *Proceedings of the ACM Symposium on InformAtion, Computer and Communications Security (ASIACCS'06)*, pp.66–77, 2006.

[38] M.P. Machulak and A. Van Moorsel, "Architecture and Protocol for User-Controlled Access Management in Web 2.0 Applications', *Proceedings of the First Workshop On Security And Privacy In Cloud Computing*, Genoa, Italy, pp.62–71, 2010..

[39] H. Hu, G. J. Ahn, and K. Kulkarni, "Ontology-based policy anomaly management for autonomic computing," In Proc. of the 7th International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom2011), USA, 2011.

[40] Y. Kouki, T. Ledoux, and R. Sharrock, "Cross-layer SLA Selection for Cloud Services," In Proc. of the First International Symposium on Network Cloud Computing and Applications (NCCA), pp. 143-147, 21-23 Nov. 2011.

[41] L. Pan, "Towards a ramework for automated service negotiation in cloud computing," In Proc. of the 2011 IEEE International Conference on Cloud Computing and Intelligence Systems (CCIS), pp. 364-367, 15-17 Sept. 2011.

[42] Y. Zhang, "An access control and trust management framework for loosly-cioupled multidomain environments," PhD Dissertation, University of Pittsburgh, 2011.

[43] Y. Zhang and J. B. D. Joshi, "Access Control and Trust Management for Emerging Multi-domain Environments," In Annals of Emerging Research in Information Assurance, Security and Privacy Services, Editors: S. Upadhyaya, R. O. Rao

[44] "Core and hierarchical role based access control (RBAC) profile of XACML", OASIS Standard, [online] Available at http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-rbac-profile1-spec-os.pdf

[45] D. Wu, X. Chen, J. Lin, M. Zhu, "Ontology-Based RBAC Specification for Interoperation in Distributed Environment," In Proc. of the First Asian Semantic Web Conference (ASWC), Beijing, China, September 3-7, 2006.

[46] R.T. Fielding and R.N. Taylor, "Principled Design of the Modern Web Architecture," *ACM Transactions on Internet Technology (TOIT)* Vol. 2, No. 2, pp.115–150, 2002.

[47] B. Motik, B.C. Grau, I. Horrocks, Z. Wu, A. Fokoue, and C. Lutz, "OWL 2 web ontology language: Profiles", W3C recommendation, W3C, October 2009.

[48] I. Horrocks, P.F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean, "SWRL: A Semantic Web Rule Language combining OWL and RuleML", Technical Report, W3C, 2004.

[49] B. Motik, U. Sattler, and R. Studer, "Query answering for OWL-DL with rules", Journal of Web Semantics: Science, Services and Agents on the World Wide Web, Vol. 3, No. 1, pp. 41-60, 2005.

[50] E. Prud'hommeaux and A. Seaborne, "SPARQL query language for RDF", W3C recommendation, W3C, January 2008.

[51] http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml

[52] J.J. Carroll, I. Dickinson, C. Dollin, D. Reynolds, A. Seaborne, and K. Wilkinson, "Jena: implementing the semantic web recommendations", In Proc. of the 13th international World Wide Web conference, pp. 74-83, 2004.

[53] E. Sirin, B. Parsia, B.C. Grau, A. Kalyanpur, and Y. Katz, "Pellet: a practical OWL-DL reasoner", Journal of Web Semantics Vol. 5, No. 2, 2007.

[54] Protege, [online] Available at http://protege.stanford.edu

[55] The OAuth, [online] Available at http://oauth.net/ (Accessed 10 November 2011).

[56] The Secure Shell (SSH) Protocol Architecture, [online] Available at http://tools.ietf.org/html/rfc4251 (Accessed 10 November 2011).

[57] The AAA Authorization Framework, [online] Available at http://tools.ietf.org/html/rfc2904 (Accessed 10 November 2011).

[58] J. Karat, C. M. Karat, C. Brodie, and J. Feng, "Designing Natural Language and Structured Entry Methods for Privacy Policy Authoring", In Proc. of the IFIP TC13 International Conference on Human-Computer Interaction (INTERACT 2005), Rome, Italy, 2005.

[59] S. Ul Haq, "Utilizing RESTful APIs In The Cloud," [online] Available at http://www.cloudtweaks.com/2013/06/utilizing-restful-apis-in-the-cloud/

[60] S. C. Markey, "REST in the cloud: A primer on RESTful APIs in the cloud," [online] Available at http://www.ibm.com/developerworks/cloud/library/cl-RESTfulAPIsincloud/

[61] SPARCLE, [online] Available at http://www.research.ibm.com/sparcle

[62] XACML, [online] Available at http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacm

[63] Protocol for Web Description Resources (POWDER): Description Resources, [online] Available at http://www.w3.org/TR/powder-dr

[64] Protocol for Web Description Resources (POWDER): POWDER-S Vocabulary (WDRS), [online] Available at http://www.w3.org/2007/05/powder-s

[65] [online] Available at http://tools.ietf.org/html/draft-hammer-hostmeta-13

[66] URI, [online] Available at http://www.ietf.org/rfc/rfc2396.txt

[67] IRI, [online] Available at http://tools.ietf.org/html/draft-duerst-iri-bis-06

[68] XRI, [online] Available at http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xri

[69] XRD, [online] Available at http://docs.oasis-open.org/xri/xrd/v1.0/xrd-1.0.html

[70] RIF, [online] Available at http://www.w3.org/TR/rif-overview

[71] CNL, [online] Available at http://en.wikipedia.org/wiki/Controlled_natural_language

[72] OSGI, [online] Available at http://www.osgi.org

[73] E. J. Coyne, "Role-engineering", In Proc. ACM Workshop on Role-Based Access Control, pages 15-16, 1995.

[74] M. P. Gallagher, A.C. O'Connor, and B. Kropp, "The economic impact of role-based access control", Planning report 02-1, National Institute of Standards and Technology, 2002.

[75] K. Brooks, "Migrating to role-based access control", In Proc. ACM Workshop on Role-Based Access Control, pages 71-81, 1999.

[76] D. Shin, G.-J. Ahn, S. Cho, and S. Jin, "On modeling system-centric information for role engineering", In Proc. ACM Symposium on Access Control Models and Technologies (SACMAT), pages 169-178, 2003.

[77] P. Epstein and R. Sandhu, "Engineering of role/permission assignment", In Proc. 17th Annual Computer Security Application Conference, pages 127-137, 2001.

[78] A. Kern, M. Kuhlmann, A. Schaad, and J. Moffett, "Observations on the role life-cycle in the context of enterprise security management", In Proc. 7th ACM Symposium on Access Control Models and Technologies (SACMAT), pages 43-51, 2002.

[79] M. Kuhlmann, D. Shohat, and G. Schimpf, "Role mining-revealing business roles for security administration using data mining technology", In Proc. ACM Symposium on Access Control Models and Technologies (SACMAT), pages 179-186, 2003.

[80] J. Schlegelmilch and U. Steffens, "Role mining with ORCA", In Proc. ACM Symposium on Access Control Models and Technologies (SACMAT), pages 168-176, 2005.

[81] J. Vaidya, V. Atluri, and Q. Guo, "The role mining problem: Finding a minimal descriptive set of roles", In Proc. ACM Symposium on Access Control Models and Technologies (SACMAT), pages 175-184, 2007.

[82] J. Vaidya, V. Atluri, and J. Warner, "Roleminer: Mining roles using subset enumeration", In Proc. ACM Conference on Computer and Communications Security (CCS), pages 144-153, 2006.

[83] D. Zhang, K. Ramamohanarao, and T. Ebringer, "Role engineering using graph optimisation", In Proc. ACM Symposium on Access Control Models and Technologies (SACMAT), pages 139-144, 2007.

[84] A. Colantonio, R. Di Pietro, and A. Ocello, "A Cost-Driven Approach to Role Engineering", In Proc. 2008 ACM symposium on Applied computing (SAC'08), pages 2129-2136, 2008.

[85] H. Lu, J. Vaidya, and V. Atluri, "Optimal Boolean Matrix Decomposition: Application to Role Engineering", In Proc. IEEE 24th International Conference on Data Engineering (ICDE2008), pages 297-306, 2008.

[86] Q. Guo, J. Vaidya, and V. Atluri, "The Role Hierachry Mining Problem: Discovery of Optimal Role Hierarchies", In Proc. 2008 Annual Computer Security Applications Conference, pages 237-246, 2008.

[87] A. Ene, W. Horne, N. Milosavljevic, "Fast Exact and Heuristic Methods for Role Minimization Problems", In Proc. ACM Symposium on Access Control Models and Technologies (SACMAT), pages 1-10, 2008.

[88] J. Vaidya, V. Atluri, and Q. Guo, "Migrating to Optimal RBAC with Minimal Perturbation", In Proc. ACM Symposium on Access Control Models and Technologies (SACMAT), pages 11-20, 2008.

[89] I. Molloy, H. Chen, T. Li, Q. Wang, N. Li, E. Bertino, S. Calo, and J. Lobo, "Mining Roles with Semantic Meanings", In Proc. ACM Symposium on Access Control Models and Technologies (SACMAT), pages 21-30, 2008.

[90] I. Molloy, N. Li, T. Li, Z. Mao, Q. Wang, and J. Lobo, "Evaluating Role Mining Algorithms", In Proc. ACM Symposium on Access Control Models and Technologies (SACMAT), pages 95-104, 2009.

[91] M. Frank, D. Basin, J. M. Buhmann, "A Class of Probabilistic Models for Role Engineering", In Proc. 15th ACM conference on Computer and Communications Security (CCS), pages 299-310, 2008.

[92] M. Frank, A. P. Streich, D. Basin, and J. M. Buhmann, "A Probabilistic Approach to Hybrid Role Mining", In Proc. 16th ACM conference on Computer and Communications Security (CCS), pages 101-111, 2009.

[93] H. Takabi and J. B. D. Joshi, "An Efficient Similarity-Based Approach for Optimal Mining of Role Hierarchy", 16th ACM Conference on Computer and Communications Security (CCS)(Poster), 2009.

[94] B. Ganter and R. Wille, "Formal Concept Analysis: Mathematical Foundations", Springer, 1998.

[95] K. Deb, "Multi Objective Optimization Using Evolutionary Algorithms", John Wiley and Sons, 2001.

[96] C. Lindig, "Fast concept analysis", In G. Stumme, editor, Working with Conceptual Structures - Contributions to ICCS 2000, 2000.

[97] http://www.st.cs.uni-saarland.de/ lindig

[98] J. Joshi, E. Bertino, U. Latif, and A. Ghafoor, "A Generalized Temporal Role-Based Access Control Model", IEEE Transactions on Knowledge and Data Engineering, Vol. 17, No. 1, pages 4-23, 2005.

[99] T. Finin, A. Joshi, L. Kagal, J. Niu, R. Sandhu, W. Winsborough, and B. Thuraisingham, "ROWLBAC - Representing Role Based Access Control in OWL", In Proc. of the 13th ACM symposium on access control models and technologies (SACMAT 2008), pages 72-83, 2008.

[100] "OWL Web Ontology Language Overview", [online] Available at http://www.w3.org/TR/owl-features/

[101] RACER,[online] Available at http://www.racer-systems.com

[102] DIG, [online] Available at http://dl.kr.org/dig

[103] JENA, [online] Available at http://jena.sourceforge.net

[104] M. Knechtel, J. Hladik, and F. Dau. "Using owl dl reasoning to decide about authorization in RBAC", In Proc. of the OWLED 2008 Workshop on OWL: Experiences and Directions, 2008.

[105] L. Kagal, T. Berners-Lee, D. Connolly, and D. Weitzner, "Using semantic web technologies for policy management on the web", In Proc. of the 21st National Conference on Artificial Intelligence (AAAI), 2006.

[106] V. Kolovski, J. Hendler, and B. Parsia, "Analyzing web access control policies", In Proc. of the International World Wide Web Conference (WWW 2007), pages 677-686, 2007.

[107] W. Di, L. Jian, D. Yabo, and Z. Miaoliang, "Using semantic web technologies to specify constraints of RBAC", In Proc. of the Sixth International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT 2005), pages 543-545, 2005.

[108] N. Heilili, Y. Chen, C. Zhao, Z. Luo, and Z. Lin, "An OWL-Based Approach for RBAC with Negative Authorization", In J. Lang, F. Lin, and J. Wang (Eds.): KSEM 2006, LNAI 4092, pp. 164-175, 2006.

[109] T. Finin, A. Joshi, L. Kagal, J. Niu, R. Sandhu, W. Winsborough, and B. Thuraisingham, "Role Based Access Control and OWL", In Proc. of the fourth workshop in the The OWL: Experiences and Direction (OWLED), 2008.

[110] M. Knechtel and J. Hladik, "RBAC Authorization decision with DL reasoning", In Proc. of the IADIS International Conference WWW/Internet, pages 169-176, 2008.

[111] L. Cirio, I. F. Cruz, and R. Tamassia, "A Role and Attribute Based Access Control System Using Semantic Web Technologies", In Proc. of IFIP International Workshop on Semantic Web and Web Semantics (SWWS), pages 1256-1266, 2007.

[112] L. Kagal, T. Finin, and A. Joshi, "A policy based approach to security for the semantic web", In Proc. of International Semantic Web Conference (ISWC 2003), 2003.

[113] L. Kagal, T. Finin, and A. Joshi, "A policy language for pervasive computing environment", In Proc. of IEEE Fourth International Workshop on Policy (Policy 2003), pages 63-76, Italy, 2003.

[114] A. Uszok, J. Bradshaw, R. Jeffers, N. Suri, P. Hayes, M. Breedy, L. Bunch, M. Johnson, S. Kulkarni, and J. Lott, "KAoS policy and domain services: Toward a description-logic approach to policy representation, deconfliction, and enforcement", In Proc. of IEEE Fourth International Workshop on Policy (Policy 2003), pages 93-98, Italy, 2003.

[115] J. Bradshaw, A. Uszok, R. Jeffers, N. Suri, P. Hayes, M. Burstein, A. Acquisti, B. Benyo, M. Breedy, M. Carvalho, D. Diller, M. Johnson, S. Kulkarni, J. Lott, M. Sierhuis, and R.V. Hoof, "Representation and reasoning for daml-based policy and domain services in kaos and nomads", In Proc. of the Autonomous Agents and Multi-Agent Systems Conference (AAMAS 2003), pages 835-842, Australia, 2003.

[116] N. Damianou, N. Dulay, E. Lupu, and M. Sloman, "The ponder policy specification language", In Proc. of Workshop on Policies for Distributed Systems and Networks (POLICY 2001), UK, 2001.