國 立 交 通 大 學

電機與控制工程學系

碩士論文

針對減輕管理負擔之條件式角色存取控制設計

Design of Condition-aware Role-based Access Control to Relax Management Overheads

研 究 生：張宗堯

Student: Tsung-Yao Chang

指導教授：黃育綸　博士

Advisor: Dr. Yu-Lun Huang

中華民國九十八年六月

June, 2009

針對減輕管理負擔之條件式角色存取控制設計

Design of Condition-aware Role-based Access Control to Relax Management Overheads

研 究 生：張宗堯　　　　Student: Tsung-Yao Chang

指導教授：黃育綸　博士　　　　Advisor: Dr. Yu-Lun Huang

國 立 交 通 大 學

電機與控制工程學系

碩士論文

A Thesis

Submitted to Department of Electrical and Control Engineering

College of Electrical Engineering

National Chiao Tung University

in partial Fulfill of the Requirements

for the Degree of

Master

in

Department of Electrical and Control Engineering

June, 2009

Hsinchu, Taiwan, Republic of China

中華民國九十八年六月

針對減輕管理負擔之條件式角色存取控制設計

學生：張宗堯　　　　　　　　　　　　　　指導教授：黃育綸　博士

國 立 交 通 大 學電機與控制工程學系（研究所）碩士班

摘　　要

存取控制的基本概念在於讓管理者能簡單、直覺並有效率的進行授權。傳統的存取控制著重於轉換使用者的身分到相對應的權限，但隨著資訊系統普遍性的發展，許多應用將會牽涉到使用者所處的環境背景，僅僅根據使用者的身分來進行授權顯得在安全性和實用性上有些不足，因此在這個研究中，我們提出一個存取控制模型，可以在進行授權時將使用者背景因素納入，透過延伸角色式存取控制(RBAC)的概念為基礎，我們加入了條件要素來限制權限的給予，使用者必須同時擁有認可的身分和符合指定的條件來獲得要求的權限，並且，我們透過將條件區分成動態得和靜態的，用不同的方式來做處理，使用者提供靜態的條件給系統來直接取得權限，系統再根據安全策略的需要去確認動態的條件，已達到減少角色數目和複雜度，以及減低反覆更新使用者屬性的成本，除此之外，在某些應用中，使用者甚至不需要進行身分確認，透過我們提出的模型，可以直接根據其擁有的條件來取得權限。結合以上幾點的改進，來達到紓解管理負擔以及使得授權更為直覺的目的。

Design of Condition-aware Role-based Access Control to Relax Management Overheads

Student: Tsung-Yao Chang Advisor: Dr. Yu-Lun Huang

Department of Electrical and Control Engineering

National Chiao Tung University

Abstract

The basic concept of an access control model intends to make the authorization simple and efficient. Conventional access control mechanisms discuss the mappings of user identities to certain permissions. With the evolution of ubiquitous computing technologies, applications have become context-aware and can interact with the context information in addition to the user identities. In this research, we propose a distinctive access control model that cooperates with context information. Our model is extended from the Role-base Access Control (RBAC) by adding a new component called "Condition", which is comprised of context information and user operations. Conditions can be treated as constraints or criteria when assigning roles and permissions to users. Different from the other models, the proposed model partitions conditions into immutable and mutable conditions, and manages different types of conditions in different ways. Users need to provide immutable conditions to obtain permissions from an access control system, while the system checks mutable conditions without user involvements. Compared with the existing models, such a design can reduce the number of roles defined in the system. By reducing the number of roles and the times required to obtain and check the conditions, we show that our model can reduce more authorization and management overhead than the existing models.

謝誌

這篇論文及其口試為我的碩士班生活畫下了完美的句點，從尋找方向、確定題目、提出構想到付諸實踐，這個過程中多少會有遇到阻礙的時候，此時接受了許多方面的協助才使得我的研究能順利的完成，其中最重要的就是黃育綸老師提供了很多建議，時常會有思考不周詳的地方需要老師幫忙釐清，寫作方面老師也花了很多時間使我的文章更為通順，真的很感謝老師，另外實驗室的黃詠文學長也提供了很多寶貴的經驗以及許多技術上的指導，沒有他的幫助我的研究肯定會坎坷不少，接下來要感謝實驗室中同屆同學們，互相討論並給予了很多寶貴的意見和經驗，值得一提的，RTES Lab所有的同學，提供了一個非常良好的學習環境，不僅僅在研究學習上，也是我在很多方面成長很多，使我的研究所生活更加精彩，最後特別高興能有家人和朋友們全力的支持，讓我能沒有後顧之憂的專心於學習上，未來並也將利用所學以及所得到的經驗，期望能有更好的表現，不辜負大家對我的期待。

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

Nowadays, computer system becomes more pervasive in home and at work. While computer systems provide the convenience of our daily life, it also brings up the concern for security requirements. In many applications, people can access to some sensitive information or privileged services in many manners [1]. So we need a way to make sure that only authorized people can access to these resources without dealing with burdensome security policies. Therefore, some access control mechanisms have been applied like Role-based Access Control(RBAC) or Access Control Lists(ACL). Those mechanisms are used to simplify the authorization for better deployment of applications.

With the development of ubiquitous computing technology, application becomes context-aware and can interact with the context information such as environment status or user activities. With people can access to the resources almost at anytime from anywhere with the Internet, the context may be essential information which characterizes the entity and some concepts are proposed [2]. For a security perspective, the major principle while processing access control becomes not only "Who they are" but also "Where they from" and "When it happened". In addition to identity, there are other contexts may affect the access control decisions. For example, in a company, a night shift staff may not open the gates in the daytime, a junior engineer can only access to the database inside his office. This context information such as location, time, type of device or connection must be took into account while processing access control.

Conventional access control mechanisms usually make the access control decisions only by the user identity. This causes a major defect because the user with different context like in different places or at different time must be distinguished. In the ubiquitous computing environment, without considering context information the access control mechanism may compromise security. Conventional access control mechanisms are too static and are not suitable to model the situations which involve context information. Thus, we need an access control mechanism that makes access control decision with the context information.

Context-aware access control has been applied in various ways [3] [4]. Sandhu et al. proposed the Role-based Access Control(RBAC) [5] [6] [7] [8] model which lets the administrator predefine the permission to role and simplify the authorization. Taking the advantage of RBAC, Covington et al. introduced Generalized Role-based Access Control(GRBAC) [9] [10] [11]. The GRBAC incorporates the concept of environment role which captures environment information and take it as constraints while permissions are being assigned to roles. Ray et al. then proposed a Location-aware role-based Access control model(LRBAC) [12] [13] which extends the RBAC model to includes location which can be treated as an context. It takes location as constraints while roles are being assigned to users. Those solutions all are based on the concept of roles and environment information [14]. But sometimes access control decision can be made only by the environment information [15]. Thus, we don't have to assign this kind of permissions to every role. Dividing Roles and Conditions to two separating authorization entry can simplify the access control policy. Besides, the context includes not only environment information but also the interaction within users. For examples, a member of a conference can only access to the conference database during the conference period, in the site of conference and with other members nearby. Furthermore, he need the authorization from the presenter to access some particular articles. This scenario shows that the access control decision can be affected by the activities of other users. And we can treat it as a context too.

Another part we are interested in is the management overheads while proceeding access control. Management overheads indicate the efforts taken by the administrator. While authorizing, less works need to be done less management overheads are taken. Those works include establishing the security policies previously according to the applications and maintaining the access control decision to be made by accurate information during a session. So we define two factors of management overheads that we need to pay attention which are the number of authorization entries and the frequency of verifying context information. ACL authorizes by listing every permission corresponding to every user. With the number of users and permissions increasing, the administrator may need lots of efforts to accomplish it. RBAC characterizes users by their capacity in the organization to reduce the number of entries. With context taking a part in access control, the relations between those entries become more complicated. LRBAC creates roles for every context information which interact with roles, yet increasing the number of roles. In GRBAC, the permissions which depend on context information are not pre-defined. Instead, the assignment varies dynamically. If the context is mutable, the ongoing access must continually updates user context and introduces extra communication overheads. But sometimes not every context varies during access time period and we don't need to update them frequently. We separate those immutable contexts between mutable ones and present our own model.

1.2 Contribution

In this research, we present an access control model that includes conditions as an entity which characterizes users. Conditions includes environment information and interaction between users and can be extended to other constraints which are similar to Roles. Our model illustrates several principles in authorization.

- Who can access What.
- Who with specific attributes can access What.
- Under what circumstances that a resource can be accessed.
- How long/much/often of a user's access
- Quality of access

Extending from RBAC, our model simplifies the authorization by the Roles. Our model can also support several basic authorization relations such as Role-Hierarchy and Separation of Duties. Also, our model introduces a way to describe how to authorize only by context and how the user interaction affects the access control decision. Most important, we separate mutable and immutable conditions to relax management overheads. Our model was presented by eXtensible Access Control Markup Language(XACML) to make it can be communicated around the cyberspace and anticipate it to adapt in many applications.

1.3 Synopsis

The thesis is organized as follows. We review the previous proposals of access control and context management in Chapter 2. Then we give the definitions of our model and expressed it with XACML in chapter 3. Chapter 4 and Chapter 5 illustrate a case study of those works on context-aware access control and compare them with our proposed model for the pros and cons. Finally, we conclude the thesis in Chapter 6.

Chapter 2

Related work

This research focuses on modeling more access control relations to cater more sophisticated applications and adapts from the NIST RBAC model. Still, there are other works that have been done to model context-aware access control in different ways. We are going to go through these previous researches here. Besides, the architecture of context-aware access control also includes the examination and process of context information. We also have a brief introduction.

2.1 Role-based Access Control

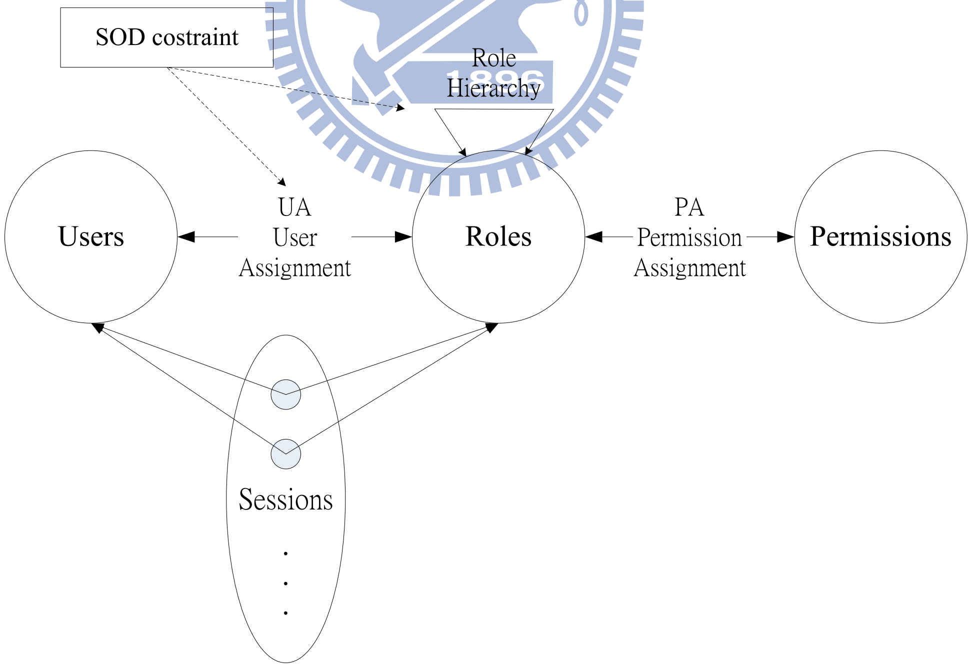

Figure 2.1: NIST Role-based Access Control Model

The RBAC model originally proposed Sandhu et al. in 1996 and was published by NIST in 2001 [5] [6] [7] [8]. The RBAC model is defined in terms of the functional roles in the organization, and then appropriately assigns users to a role or set of roles. There are several parts we need to pay attention traditionally which are Users, Roles, Objects, Permissions and Sessions. User indicates a remote device or a human being for simplicity. Role is a named job function within the organization and can be treated as a set of authorities and responsibilities. Object can be data or resource that provided by the system. Permission is a clearance of access to one or more objects. Session allows user to active a subset of the roles they belong to. Beyond this scenario, several relations can be established which are user-role relation, role-permission relation and role-role relation. With RBAC, role-permission relations can be predefined. Administrator can easily confer and revoke users to roles instead of assign permissions directly to users. RBAC also supports hierarchical roles for advance usage. The main security principles of RBAC are listed below.

2.1.1 Least Privilege

Only the specified permission sets can be assigned to the role. This feature simplifies the relation between Permissions and Users.

2.1.2 Separation of Duties

For sensitive tasks, the functions of Roles may not be assigned to the same Users at the same time. For example, if a bank account holder and also be a teller who transact his account, it may cause some security problems. In this scenario, we need the functions of Roles be mutually exclusive. We can define two varieties of Separation of Duties(SOD) which are dynamic and static. For dynamic SOD, Users can not activate those Roles which are mutually exclusive at the same time. Then for static SOD, those Roles which are mutually exclusive can't be activated even assigned to the same Users. RBAC supports these kinds of constraint.

2.1.3 Data Abstraction

Administrator can establish abstract permissions other than typical permissions like read, write and execute.

2.1.4 Role Precedence

Sometimes, confliction may occur between two different Roles which are belonged to the user session. An operation which is allowed by one Role may be denied by another. This problem can be solved by Separation of Duties, but we also can establish some algorithm in this case to make it simple. The simplest way is make the access control decision always be allowed or denied.

2.1.5 Role Hierarchy

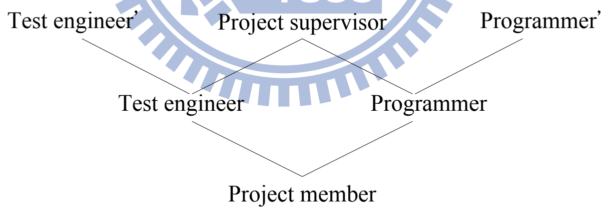

Figure 2.2: An Example of Hierarchical Roles

In most organizations, the positions or job functions have hierarchical relations. While constructing Roles, we can take the advantage of these relations to simplify the privilege. A role which inherits another's permissions is called Senior Role, and the role which is inherited is called Junior Role. RBAC supports these kinds of inheritance and let the administrators manage role more easily. Overall, the user acquires a set of roles according of its identity and invokes a session of the roles activated. The permissions which assigned to these roles can be easily obtained by previous definition then determine whether the user can takes those operations.

2.2 Generalized Role-based access control

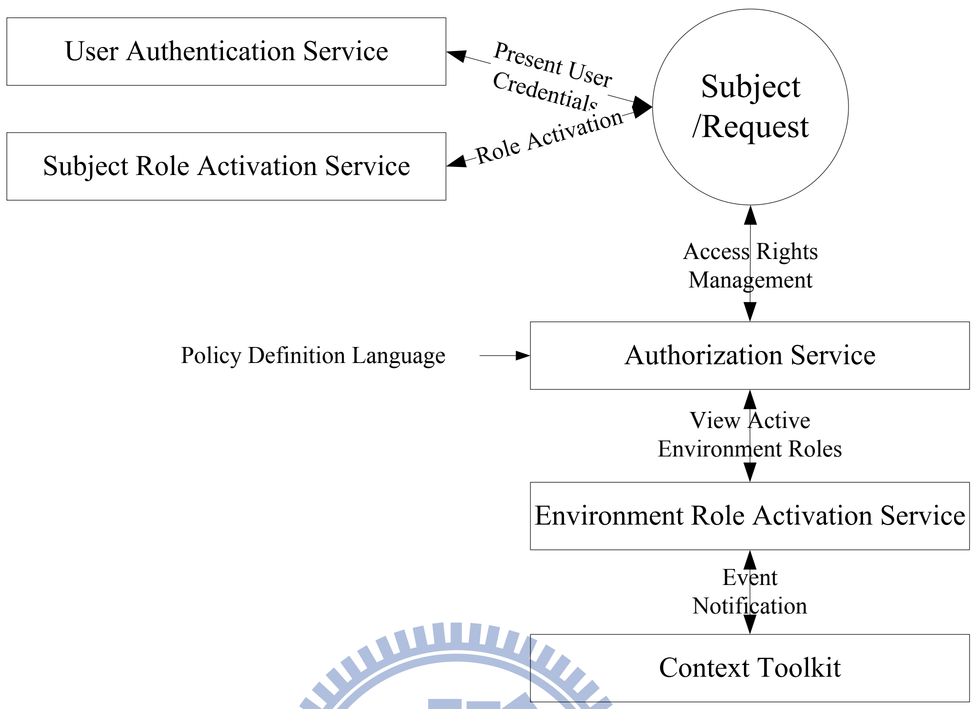
User Authentication Service
Present User Credentials
Subject Role Activation Service
Role Activation
Subject /Request
Access Rights Management
Policy Definition Language
Authorization Service
View Active Environment Roles
Environment Role Activation Service
Event Notification
Context Toolkit

Figure 2.3: Generalized Role-based Access Control Model

Covington et al. introduced GRBAC [9] [10] [11]. The Roles in GRBAC are defined as a group of users other than a set of permissions. Similarly, the Environment Roles are a set of environment statuses which can be constructed hierarchical. For a User accessing an object, he should be assigned a Role and there are some transactions that allow this Role to access this object when the current Environment Role is active. In this case, the Environment Role acts as an objective background and should be checked frequently. As long as the environment statuses change, the system makes another authorization without the user request.

2.3 Location-aware Role-based access control

Ray et al. proposed LRBAC model [12] [13]. LRBAC model extends RBAC to incorporate location information by adding a new component location into RBAC model. Locations are associated with roles. Roles can be assigned or activated in some particular location. Thus, each

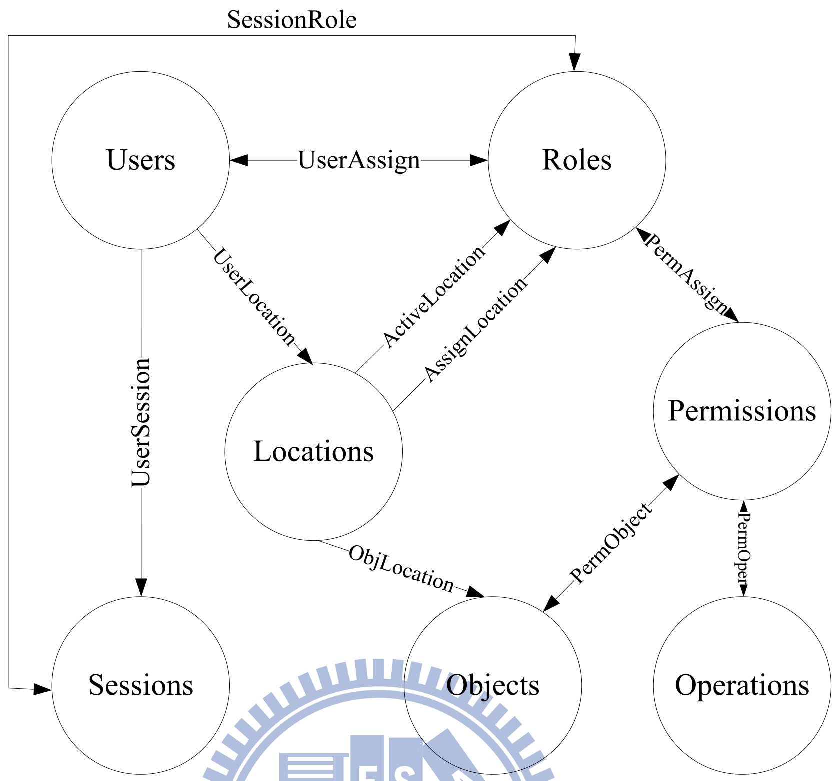

Figure 2.4: Location-aware Role-based Access Control Model

valid role is associated with a set of locations where the role can be assigned or activated. A user is allowed to perform an operation on an object, if it has been assigned a role which includes a permission of the operation on the object. Moreover, the user location must be included in those locations associate with the role which make it can be assigned or activated.

2.4 Context Verification

In general, context information is often publicly and freely available like time and location. But we still need a method to determine if the information meets our requirements. At the same time, we need to make sure the context information we attained is reliable. In order to do so, there are several criteria with the context verification.

- **Ensuring the source**. The context verification can be performed by certain entities such as the user, the system or a trusty third party. Either one is accessible depending on the applications. But for the security concern, performing by the system or a trusty third party is preferable to by the user.

- **Recording user context history**. In the physical world, there are some restrictions on user activity. By comparing the user context information to its history or to other user context information, we can find out some presentation is not reasonable. For example, user location can't change in an irrational rate or path. In some places, distance between two users can't be too close or too far even overlap. This restrictions all can help us to verify the user context information.

- **Ensuring user privacy**. While recording the user context information, it comes out the issue of user privacy. In some applications, context may be sensitive information. So we need to protect this information from other irregular usage.

2.4.1 Location Information

In many applications, the location becomes essential information for variety of purposes [16]. User may request the rights to access some information from many different places. The assumption that user identity fully determines what they are authorized to do is not sufficient. So location is major context information that we focus on.

For some top secret information, it requires having a physical presence in the room where the system locates. But this scenario needs to isolate this top secret information from those usual information on different system. This makes a lot of inconvenient. According to the assumptions, a top secret place only allows people with security clearance to enter. Also, top secret information will only be contained in a top secret place. These two intuitive concepts enforce the security policy. By verifying the user location, the user privilege can be determined at the same time. In order to achieve the location verification, the software applications must be reachable via a telecommunication infrastructure. Fortunately, new and enhanced location determination techniques allow us to get location information more accuracy.

Location Determination

According to current location determination method, Global Positioning System(GPS) is the most popular one. But it can only take the advantage on the outdoor environment. For indoor location determination, Location determination techniques can be classified into range-dependent and

range-independent ones [17]. Range-dependent location determination indicates that we need to measure the range between nodes to determine the location of the device. The infrastructure includes several reference points which are disposed at proper positions. Usually, these techniques rely on measuring specific properties of the exchanged signals, such as the received signal strength(RSS), Time of Arrival(ToA), Angle of Arrival(AoA) and the Time Difference of Arrival(TDoA). Those properties depend on the distance between nodes and can be used to measure the distance between them. Range-independent location determination, instead of measuring the distance, uses other characteristics to determine proximity. For example, it compares the signal properties from defferent reference points or to the data base, or simply identifies its location through physical contact with other nodes. In this research, we assume that the location verification can be made by either technique stated above.

Location Representation

By today's technology of mobile network, it's hard to ensure fully exact user location. It also costs more to determine the absolute location. Instead of measuring the user's absolute location, it's more sufficient to verify if the user location meet certain criteria. Ardagna et al. proposed location verification which makes location query while user requests for object [18] [19]. Because of the interoperability of location determination, we separate the location determination from the access control engine and call it location service, the access control engine ask the location service if the user is in the appropriate place. The query includes several kinds of representation depending on the access control policies, such as within area, velocity, density, etc. And the location service responds this query in a binary response with a confidence value. The access control engine receives the information to make the authority. By recording a series of confidence value of different location services, the access control engine can set the Service Level Agreement(SLA) of the location service to decide the information from certain location service to be credible or not. This concept is proposed to compete the uncertainty of the location determination.

2.5 eXtensible Access Control Markup Language(XACML)

XACML is a declarative access control language implemented in Extensible Markup Language(XML). Latest version XACML 2.0 was ratified by OASIS standards organization in 2007 [20] [21]. XACML defined a core schema and corresponding namespace to express the authorization policy in XML. It makes the access control easier to share between applications and communicate with other. Anderson express RBAC with XACML in a formal way [22] and we extend it to express our model.

Chapter 3

Condition-aware RBAC Model

This section explains a typical framework of an access control system, followed by the proposed condition-aware RBAC model.

3.1 Overview

Mainly, we can divide an access control system into three parts. First, the certificate authority is a trust third party that gathering information about users then transforms it to certain certificate in order to proof the identities or attributes of users. Second, the access control decision point receive the certificate provide by the authorities and determine if the user can take the requesting operations according to the access control policy. Finally, the access control decisions are sent to the access control enforcement point which actually implement the access control and provide the resource. The framework is like Fig 3.1. Our model focuses on how the access control decisions are made. We assume that certificate authorities are reliable and can accept frequently request on user conditions. Then our model acts as the access control decision take these information and proceed the authorization.

There are two ways access control decision point receiving the user certificate: push mode and pull mode. In the push approach, user presents its identity and condition certificates to the access control decision point. In the pull approach, access control enforcement point gathers related information of user. Considering the continuity and mutability of conditions, it is not

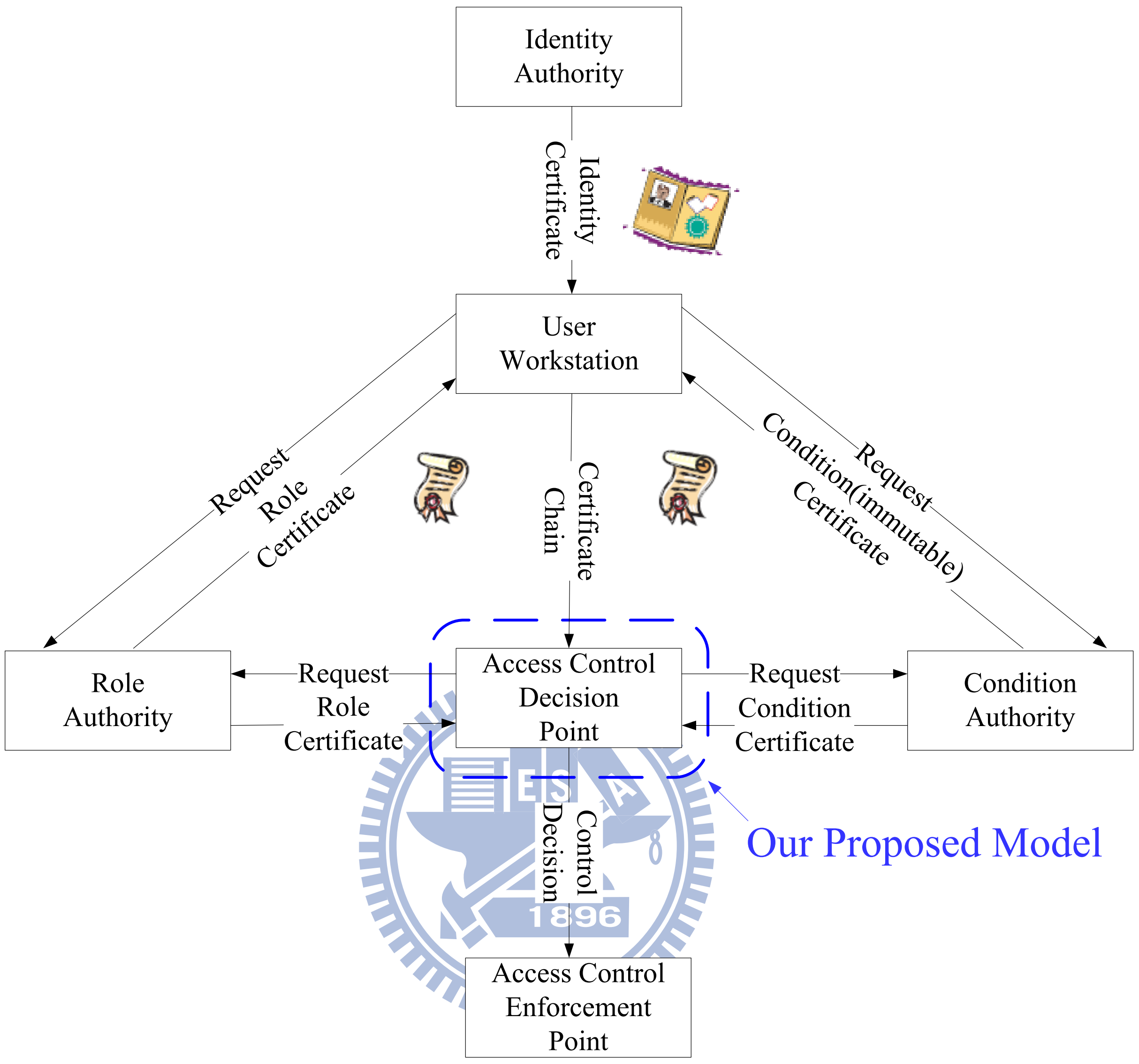

Figure 3.1: System Framework Overview

efficient and scalable to apply only either of one approach. In our proposal, we separate the immutable and mutable conditions to apply both push and pull mode respectively. We'll discuss this concept in the upcoming sections.

3.2 Proposed Model

Our proposed model is a distinctive architecture for context-aware access controlling. We give out an intuitional view and meaningful relations between the Users and Permissions. Administrators need not deal with burdensome access control policies and directly reach the fruitful result which is a security and lightweight access control mechanism.

Our model as Fig 3.2 shows how the access control decisions are made. There are four major components which are Users, Roles, Conditions and Permissions. Users indicate a human being. Roles are the job functions which basically are related to the capacity in the organization. Conditions mean the context of Users who make the access request. And Permissions stand as the approval to access some resources. Our model includes two ways to authorize Users which are Roles and Conditions. The main goals are simplifying the relations between Permissions and Users and making the authorization more related to the real framework of the organization. In many cases, Roles and Conditions may interact with each other. Our model also takes this constraint into account. Furthermore, this kind of interaction may also present between certain Users. Our model clearly describes these relations. We will discuss each relation in the upcoming chapters more detailed.

3.2.1 Design Issues

We start with the RBAC model. The basic concept of RBAC is that multiple users and permissions associate with some particular role. Users acquire permissions by belonging to the same role with the permission. Basically, there is no constraint on these relations. LRBAC introduces location as a constraint while users are being assigned to roles. In this research, instead of a constraint, we treat conditions as sets of permissions like roles. Permissions are directly assigned to conditions. Conditions can be the only characteristics of User which determine authorizations. Stills, Conditions can also work with the user Roles. But now we first describe how we make access control decision by Conditions.

In some applications, access control decisions could rather preferably made by the users'

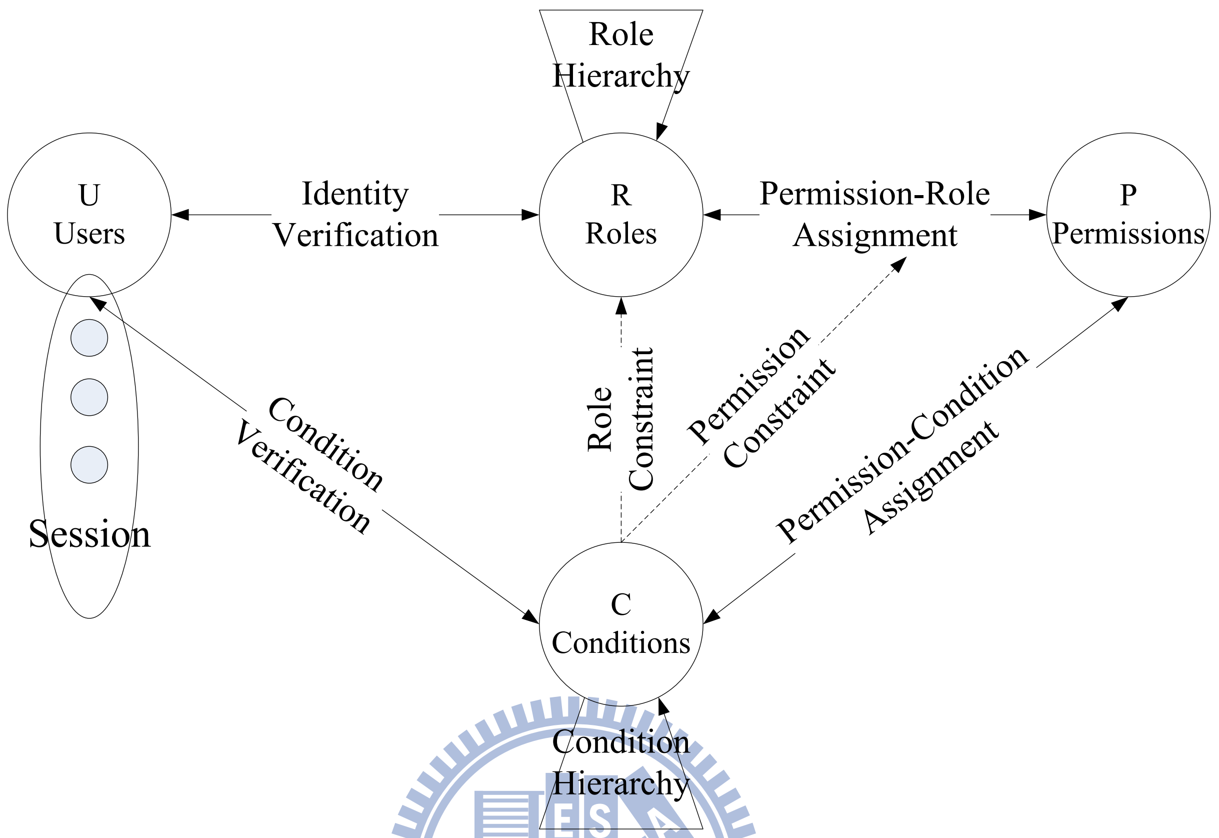

Figure 3.2: Our model

contexts than on their identities. For example, in MRT system, the payments of the passengers mostly depend on the stations they got in and out. In this case, we don't really care about who the passengers are, but we are interested in how long they traveled and if we let them leave or not. Besides, some social network platforms or forums may support anonymous using. But it still has some Conditions which limit Users authorities. Such scenario, we need an access control mechanism that directly couple Users and Permissions by Conditions.

In terms of the applications, the Conditions may have several properties.

- Assigning Conditions to Users vary more frequently than assigning Permissions to Conditions.
- User context also varies more frequently than its identity.

- Conditions can be hierarchical in many ways.
- A user can only be assigned one content for one particular condition.

Comparing those properties to the definitions of Roles in RBAC, it's reasonable for us to establish Conditions as a component like Roles as shown in Fig 3.3. The authority to some particular Conditions are predefined. Then Users grant their status of Conditions though condition verification. As long as the system verifies the Conditions, Users grant the corresponding Permissions. For example, in a library, some publications are only allowed accessing inside the building. The permissions for accessing this information can be assigned to the Condition *in the library*. While users are verified being in the library, they are able to access this secure information.

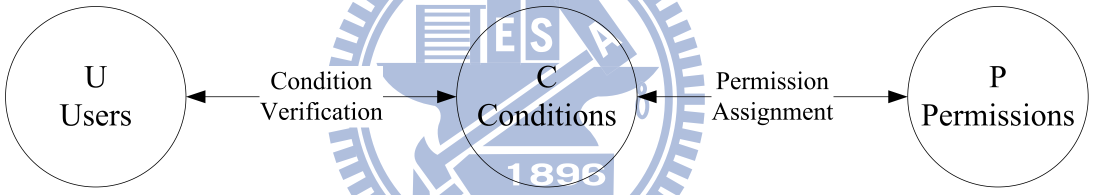

Figure 3.3: Relations between Users, Conditions and Permissions

Next, considering the fact that Users can also be assigned Roles by their identity, we combine those two methods as in Figure 3.4. In short, Users can grant Permissions by their Roles or Conditions if the security policies only require either one. As Roles and Conditions are sets of Permissions and can be predefined, our model can achieve Least Privilege and reduce the burden on the administrator.

Overall, our access control model becomes Fig 3.2. As mention above, Users can be authorized to take some operations by Roles or Conditions separately. Besides, Conditions may affect the access control decision, by being constraints on assigning Roles to Users or on assigning Permissions to Roles. For the same constraints, Conditions can act in either one way most of the time. Complying with the basic concepts of RBAC, we try to make Users acquiring

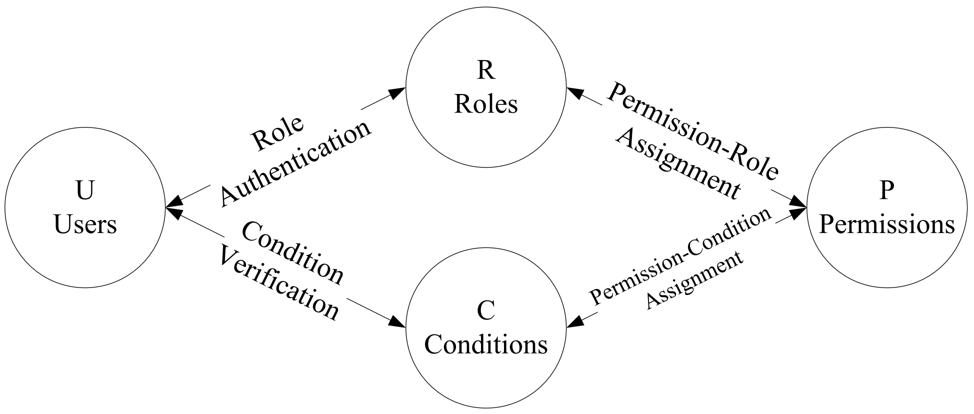

Figure 3.4: Combination of Role-based and Condition-based Authorization

Permissions directly by their Roles. Comparing the access operation time period to the varying frequency of Conditions, we can separate those mutable Conditions from the immutable ones. In order to make the relations between Users and Roles being stable, we make those immutable Conditions as constraints to Roles from being assigned. This kind of Roles is related with some Conditions. Users push their certificate on identity and immutable Conditions with the access request. Only the Users who are verified to have those Conditions can be assigned such Roles. For example, "night-time" is an immutable Condition. The Role "night-shift" is related with the Condition "nighttime ". Users can only acquire the Role "night-shift" with the Condition "night-time ". And for those mutable Conditions, we take them as constraints on assigning Permissions to Roles and they need to be verified by frequently pulling the update on mutable Conditions. For example, "In the building" is a mutable Condition. The night-shift staff can access the wireless network only inside the building. So the Role "night-shift" must be verified Condition "In the building" to activate the Permission "Wireless Network".

Session is another important component in our model. A user may have multiple certificates that allow him acquire multiple roles and conditions. Those roles and conditions form the user session. User can decide which roles or conditions he wants to activate during current access. As long as the activating roles and conditions complying with the concept of Separation of Duties or other constraints, user can get all the permissions that assigned to those activating roles and conditions. Next, we discuss about the extra feature in our model.

Separating Immutable and Mutable Conditions

Every ongoing access has its access time period. Suppose it requires several Conditions. Some of them may not change during once access. For example, entering the gate may not last over tens of seconds. If a staff only be permitted entering the gate during weekday. The Condition "Weekday" may not change during such staff entering the gate. And we call this kind of Conditions immutable conditions. And for the mutable conditions which may vary during the ongoing access, any change of these mutable conditions in concurrent access should invoke the updating.

Approval Confirmation

Relatively Roles can also affect Conditions from being assigned. Our model describes the relation that other user's previous operations may affect current authorization. For example, for a shared data, an ongoing write operation is performed. Current user can't acquire the write permission at the same time. Or a security policy needs another user's approval to be confirmed. We treat this previous operation as a Condition that approves certain permissions to be available. This kind of Conditions related with certain Roles. For example, entering the laboratory needs the approval from the professor. So the Roles "professor" decides if the Condition "Approval-on-lab" can be assigned. Users need this Condition "approval from the professor" to enter the laboratory.

Separation of Duties(SOD)

While other user's operations may affect certain permissions on being assigned, the SOD becomes more important. Traditional SOD describes the situation that a user can't be a player and a referee at the same time. Our model retains this constraint and introduces conditions into this constraint. We can describe certain roles or conditions can't be assigned to or activated by the same user. In the other word, under different conditions, different sets of roles may be exclusive to each other.

Condition Hierarchy

With Role Hierarchy, a senior role can inheritance the permissions which assigned to a junior role beyond its capacity. Similarly, many conditions also have hierarchical architecture like location and time. Then we illustrate a combination of Role and Condition Hierarchy. We treat a role with condition as a senior role. We use an example to explain this relation.

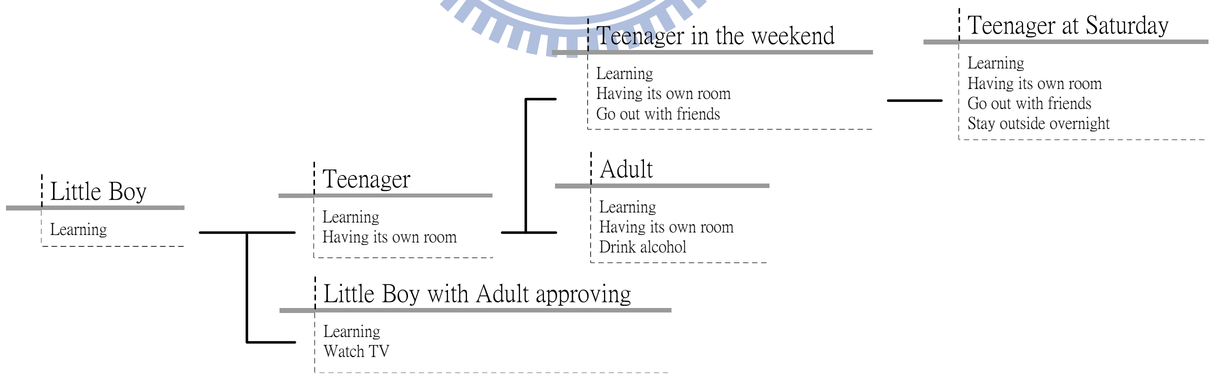

Figure 3.5: Example of Hierarchical Conditions

As in Fig 3.5, the relations between Role "Little Boy", "Teenager" and "Adult" form a Role Hierarchy. And Condition "Weekend" and "Saturday" form a Condition Hierarchy. While we treat them as immutable conditions and construct roles for them, Role "Teenager in the weekend" becomes a senior role to Role "Teenager". Similarly, Role "Little Boy with Adult Approving" inheritances the permission assigned to Role "Little Boy".

The relations above provide several ways of authorizing and constraining on the applications which involve context information. Finally, our model becomes like Fig 3.6 and we define it in the next section.

3.2.2 Model Definitions

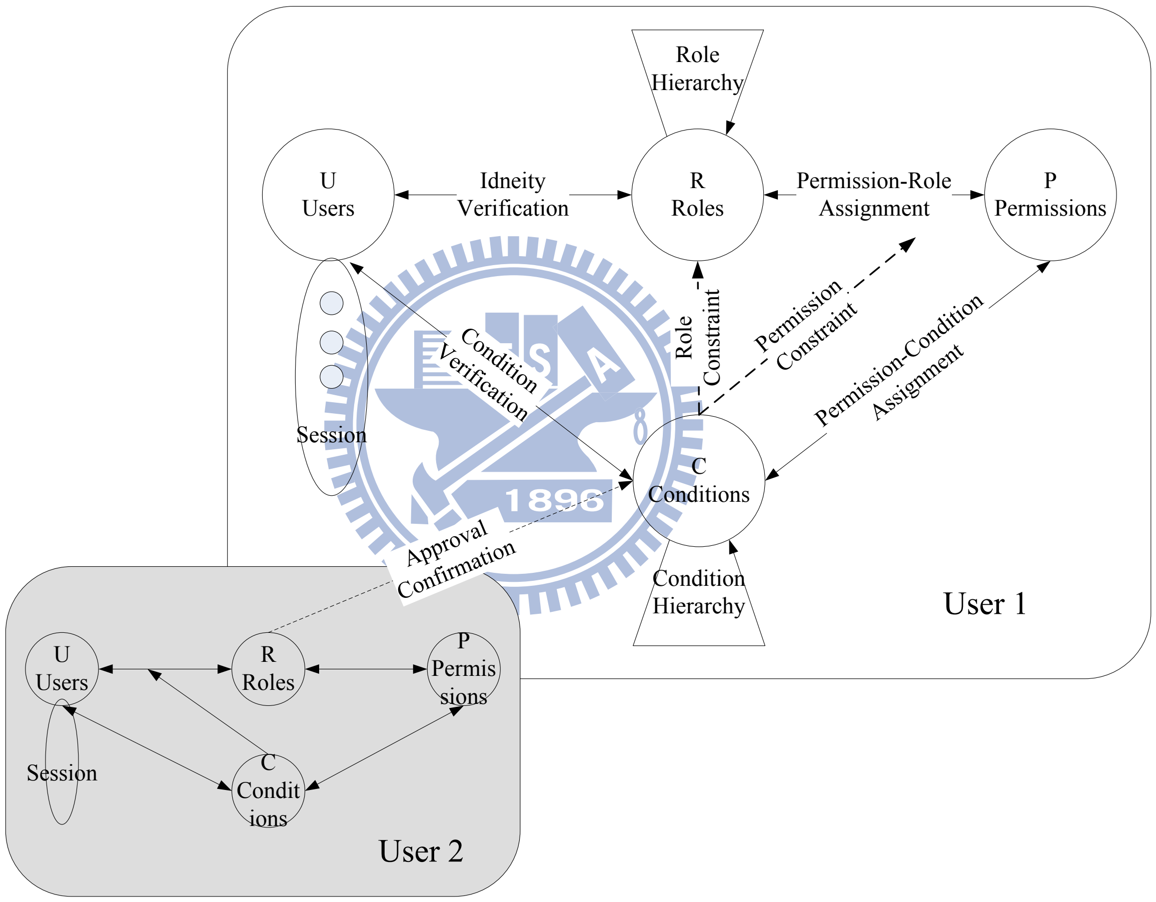

Figure 3.6: Our model with Approval Confirmation

In our access control as Fig 3.6, the major components are described as below.

U: Users Users indicate human beings or other autonomous agent such as a process or a device. We assume that each User requests some system resource through a remote device and the device can be detected by the condition service.

P: Permissions Permissions are approvals of taking an operation to some particular resources. These operations includes typical operations like read, write and execute and other operations depend on the application.

R: Roles Roles are sets of Permission and often related to the capacity in the organization.

S: Session A user can be assigned multiple Roles and Conditions at the same time. And those Roles and Conditions form the user Session. During the authorization, Users can decide to activate certain Roles or Conditions to proceed.

C: Conditions Conditions are the factors of environment the actions took by other Users which may affect the access control decisions.

Next, we describe the associations between the above components.

UA: User Assignment UA includes Identity Verification and Condition Verification. Both IV and CV are used to mapping Users to certain permission sets.

- **IV: Identity Verification** IV is the mapping that assigns Roles to Users.
  After authenticating, Users are classified to some Roles by their identity. A user can be assigned with a set of Roles called Session. While making access requests, User decides to activate the Roles in their Session to acquire certain Permissions.
- **CV: Condition Verification** CV is the mapping that assigns Conditions to Users.
  The system grants user attribute by the condition service. The Condition Verification includes location determination, time system, environment sensor or approval from other Users. Those depend on the application requirements.

Role Constraint Some Conditions should be granted to acquire particular Permissions. For the immutable one, Roles assigned these Permissions are associate with these immutable

Conditions. Users need to grant these immutable Conditions to acquire these Roles. This relation is Role Constraint.

PA: Permission Assignment PA includes Role Assignment and Condition Assignment. PA maps Roles and Conditions to certain predefined Permissions.

- **RA: Permission-Role Assignment** RA is the mapping that assigns Permissions to Roles.

  Permissions associate with roles previously according to the application rules.

- **CA: Permission-Condition Assignment** CA is the mapping that assigns Permissions to Conditions.

  Permissions also associated with conditions. A condition can be assigned with several permissions.

Permission Constraint As mentioned above, some Conditions are relative mutable and be a constraint on assigning Permissions to Roles. Conditions need to be verified frequently to allow User acquiring these Permissions, even if User already grant the corresponding Roles.

Approval Confirmation The approval is the authorization proof from other Users or we can say from other Roles. We treat the approval as a Condition that includes certain Permissions. The Approval Confirmation allows these Conditions to be acquired.

Then we illustrate the definitions of our model.

- U, R, C, P, represent sets of Users, Roles, Conditions and Permissions respectively.

- $UA = IV \cup CV$

- $IV \subseteq U \times R \times C$, the policy that maps Users to Roles.

  $assignedUsers(r : R) \rightarrow 2^U \equiv assignedUsers(r : R) = \{u \in U \mid (u, r) \in UA\}$

  With the effect of Role Constraint, the association between Users and Roles may takes Conditions into account and becomes the statements below.

  $assignedUsers(r : R, c_i : C) \rightarrow 2^U \equiv assignedUsers(r : R, c_i : C) = \{u \in U \mid (u, r, c_i) \in UA\}$

- $CV \subseteq U \times C$, the policy that maps Users to Conditions.

  $assignedUsers(c : C) \rightarrow 2^U \equiv assignedUsers(c : C) = \{u \in U \mid (u, c) \in CV\}$

• $PA = RA \cup CA$

- $RA \subseteq R \times P$, the policy that maps Roles to Permissions.

  $assignedPerms(r : R) \rightarrow 2^P \equiv assignedPerms(r : R) = \{p \in P \mid (r, p) \in RA\}$

  With the effect of Permission Constraint, the association between Permissions and Roles may takes Conditions into account and becomes the statements below.

  $assignedPerms(r : R, c_i : C) \rightarrow 2^U \equiv assignedPerms(r : R, c_i : C) = \{p \in P \mid (u, r, c_i) \in RA\}$

- $CA \subseteq C \times P$, the policy that maps Conditions to Permissions.

  $assignedPerms(c : C) \rightarrow 2^P \equiv assignedPerms(c : C) = \{p \in P \mid (c, p) \in CA\}$

• The statements above define the associations that a user can have within the model. With the effect of Session, a user can have multiple associations with Roles and Conditions as defined below.

- $sessionUser : S \rightarrow U$, the association that maps Session to the single user they belong to.

- $sessionRoles : S \rightarrow 2^R$, the association that maps Session to the set of Roles.

  $sessionRoles(s) \subseteq \{r \mid (sessionUser(s), r, c) \in UA\}$

- $sessionConds : S \rightarrow 2^R$, the association that maps Session to the set of Conditions.

  $sessionConds(s) \subseteq \{r \mid (sessionUser(s), c) \in CV\}$

- $sessionPerms : S \rightarrow 2^P$, the Permissions this Session has.

  $sessionPerms(s) = \cup_{r \in sessionRoles(s)} \{p \mid (r, p) \in RA\}$

3.2.3 Examples

Next, we illustrate an example of our model. Suppose that a night shift staff can acquire the right to access the company gate at nighttime. Also, he can use the wireless network while being inside the building. The authorization process is stated below.

- The authentication and condition service verify his identity and conditions.

- The access control engine receives the access request like $\{U_{id}, Context_1, Context_2, ...\}$. In this case, $Context_1$ may stands as location amd $Context_2$ as time.

- The access control engine assign Roles and Conditions to user based on the information in the access request and the access control policies. In this case, the user may be assigned the Role "employee", "night-shift" and the Condition "In-the-building" and "nighttime", then the access control engine makes the access control decisions. Since Condition "in the building" is mutable relatively and Condition "night-time" is immutable. The user Session becomes as below.

  - $r_1 : employee.r_2 : night - shift, c_1 : In - the - building, c_2 : nighttime$

– $UA : (U_{id}, employee), (U_{id}, nighttime, night - shift)$

$RA : (night - shift, gate), (employee, In - the - building, WirelessNetwork)$

3.3 Realization

XACML is a general-purpose access control policy language and can be use to describe the policy and access control decision request and response. Applying XML makes it easier to communicate, so we extend XACML to illustrate our model in a formal way. In order to improve readability, the specification assumes use of the following XML Internal Entity declarations:

```
^lt;!ENTITY xml "http://www.w3.org/2001/XMLSchema#">
^lt;!ENTITY rule-combine
"urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:">
^lt;!ENTITY policy-combine
"urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:">
^lt;!ENTITY function "urn:oasis:names:tc:xacml:1.0:function:">
^lt;!ENTITY subject-category
"urn:oasis:names:tc:xacml:1.0:subject-category:">
^lt;!ENTITY subject "urn:oasis:names:tc:xacml:1.0:subject:">
^lt;!ENTITY role "urn:oasis:names:tc:xacml:2.0:subject:role">
^lt;!ENTITY roles "urn:example:role-values:">
^lt;!ENTITY resource "urn:oasis:names:tc:xacml:1.0:resource:">
^lt;!ENTITY action "urn:oasis:names:tc:xacml:1.0:action:">
^lt;!ENTITY actions "urn:oasis:names:tc:xacml:2.0:actions:">
^lt;!ENTITY environment "urn:oasis:names:tc:xacml:1.0:environment:">
```

According to XACML, We specify the components as below.

Users are expressed by XACML Subjects.

Roles are expressed by XACML Subject Attributes.

Conditions are expressed by XACML Subject Attributes.

Objects are expressed by XACML Resources.

Operations are expressed by XACML Actions.

Permissions are express by XACML Permission <PolicySet> and linked to the corresponding roles by Role <PolicySet>.

And the associations between components are as Policy or PolicySet in XACML.

Role <PolicySet> associates user who has particular role to a Permission <PolicySet>.

Permission <PolicySet> that associates role to a set of permissions.

Condition <PolicySet> associates user who has particular condition to a Condition <PolicySet>/

Condition <PolicySet> associates condition to a set of permissions.

Role Assignment <PolicySet> specifies which roles can be assigned to or activated by which users. Role Assignment <PolicySet> is performed by Role Authority.

Condition Assignment <PolicySet> specifies which roles can be assigned to or activated by which users. Condition Assignment <PolicySet> is performed by Role Authority.

Next, we illustrate how a user grants its permissions by their roles and conditions by a simple Policy "Clerk can only enter the Gate in Weekday". First, user requests its roles and conditions from the role and condition authority. The role and condition authority decide which roles and conditions he can acquire by the Role Assignment <Policy> and Condition Assignment <Policy>. The Condition Assignment <Policy> may like the table below and similar to the Role Assignment <Policy>.

Condition Assignment <PolicySet>

```
<Policy xmlns="urn:oasis:names:tc:xacml:2.0:policy:shema:os"
PolicyId="Condition:Assignment:Policy" RuleCombiningAlgId="&rule-combine;deny-overrides">
<Rule RuleId="in:weekday:condition:requirements" Effect="Permit">
  <Target>
    <Resources>
      <Resource>
        <ResourceMatch MatchId="&function;string-equal">
          <AttributeValue DataType="&xml;string">condition;in Weekday<AttributeValue>
          <ResourceAttributeDesignator AttributeId="condition" DataType="&xml;string"/>
        </ResourceMatch>
      </Resource>
    </Resources>
    <Actions>
      <Action>
        <ActionMatch MatchId="&function;string-equal">
        <AttributeValue DataType="&xml;string">&action;enableCondition</AttributeValue>
       <ActionAttributeDesignator AttributeId="&action;action-id' DataType="&xml;string"/>
        </ActionMatch>
      </Action>
    </Actions>
  </Target>
  <Condtion>
    <Apply FunctionId="&function:and">
      <Apply FunctionId="&function:greater-than-or-equal">
        <Apply FunctionId="&function:string-one-and-only">
        <EnvironmentAttributeDesignator DataType="http://www.w3.org/2001/XMLSchema#string"
AttributeId="&environment;DayOfWeek"/>
        </Apply>
      <AttributeValue DataType="&xml;string">1</AttributeValue>
      <Apply FunctionId="&function:greater-than-or-equal">
        <Apply FunctionId="&function:string-one-and-only">
        <EnvironmentAttributeDesignator DataType="http://www.w3.org/2001/XMLSchema#string"
AttributeId="&environment;DayOfWeek"/>
        </Apply>
      <AttributeValue DataType="&xml;string">1</AttributeValue>
      <Apply FunctionId="&function:less-than-or-equal">
        <Apply FunctionId="&function:string-one-and-only">
        <EnvironmentAttributeDesignator DataType="http://www.w3.org/2001/XMLSchema#string"
AttributeId="&environment;DayOfWeek"/>
        </Apply>
      <AttributeValue DataType="&xml;string">5</AttributeValue>
  </Condition>
</Rule>
</Policy>
```

Role and Condition Authority assign roles and conditions to the user by his identify and context information. Then the user uses these certificates to acquire corresponding permissions. The Role <PolicySet> associates user to the Role "Clerk in Weekday" and the Permission <PolicySet> specifies the actual permissions it has. The Permission <PolicySet> of Role "Clerk in Weekday" also references to the Permission <PolicySet> of Role "Clerk" which allows user to inheritance the permissions assigned to Role "Clerk".

Role <Policy Set>

```
<PolicySet xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:os"
PolicySetId="RPS:clerk:in:weekday:role">
  <Target>
    <Subjects>
      <Subject>
        <SubjectMatch MatchId="&function;anyURI-equal">
          <AttributeValue DataType="&xml;anyURI">&role;clerk</AttributeValue>
          <SubjectAttributeDesignator AttributeId="&2.0:role" DataType="&xml;anyURI"/>
        </SubjectMatch>
      </Subject>
    </Subjects>
  </Target>
  <Condition>
    <Apply FunctionId="&function;string-equal">
      <Apply FunctionId="&function;string-one-and-only">
        <SubjectAttributeDesignator DataType="&xml;string" AttributeId="condition"/>
      </Apply>
      <AttributeValue DataType="&xml;string">in Weekday</AttributeValue>
    </Apply>
  </Condition>

  <PolicySetIdReference>PPS:clerk:in:weekday:role</PolicySetIdReference>
</PolicySet>
```

Permission <PolicySet>

```
<PolicySet xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:os"
PolicySetId="PPS:clerk:in:weekday:role" PolicyCombinungAlgId="&policy-combine;deny-overrides">
  <Policy PolicyId="Permissions:specifically:for:the:clerk:in:weekday:role
RuleCombiningAlgId="&rule-combine;deny-overrides">
    <Rule RuleId="Permission:enter:the:gate" Effet="Permit">
      <Target>
        <Resources>
          <Resource>
            <ResourceMatch MatchId="&function;string-equal">
              <AttributeValue DataType="&xml:string">Gate</AttributeValue>
                         <ResourceAttributeDesignator  AttributeId="&resource;resource-
id" DataType="&xml;string"/>
            </ResourceMatch>
          </Resource>
        </Resources>
        <Actions>
          <Action>
              <ActionMatch MatchId="&function;string-equal">
                <AttributeValue DataType="&xml;string">enter<AttributeValue>
           <ActionAttributeDesignator AttributeId="&action;action-id" DataType="&xml;string"/>
              </ActionMatch>
            </Action>
          </Actions>
        </Target>
      </Rule>
    </Policy>

    <PolicySetIdReference>PPS:clerk:role</PolicySetIdReference>
</PolicySet>
```

The Policy above should how immutable condition works. For mutable conditions, the Permission <PolicySet> restricts such permissions from being acquired. The Permission <PolicySet> may like below.

Permission <PolicySet>

```
<PolicySet xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:os"
PolicySetId="PPS:clerk:role" PolicyCombinungAlgId="&policy-combine;deny-overrides">
  <Policy PolicyId="Permissions:specifically:for:the:clerk:role RuleCombiningAlgId="&rule-
combine;deny-overrides">
    <Rule RuleId="Permission:access:WirelessNet" Effet="Permit">
      <Target>
        <Resources>
          <Resource>
            <ResourceMatch MatchId="&function;string-equal">
              <AttributeValue DataType="&xml:string">Wireless Network</AttributeValue>
                     <ResourceAttributeDesignator  AttributeId="&resource;resource-
id" DataType="&xml;string"/>
            </ResourceMatch>
          </Resource>
        </Resources>
        <Actions>
          <Action>
            <ActionMatch MatchId="&function;string-equal">
              <AttributeValue DataType="&xml;string">access<AttributeValue>
          <ActionAttributeDesignator AttributeId="&action;action-id" DataType="&xml;string"/>
            </ActionMatch>
          </Action>
        </Actions>
      </Target>
      <Condtion>
        <Apply FunctionId="&function:greater-than-or-equal">
          <Apply FunctionId="&function:string-one-and-only">
                     <EnvironmentAttributeDesignator  DataType="http://www.w3.org/2001/
XMLSchema#string" AttributeId="&environment;location"/>
          </Apply>
          <AttributeValue DataType="&xml;string">building</AttributeValue>
        </Apply>
      </Condition>
    </Rule>
  </Policy>

  <PolicySetIdReference>PPS:visitor:role</PolicySetIdReference>
```

```
</PolicySet>
```

Chapter 4

Case Study

In this chapter, we illustrate a case study about access control to see how previous model and mine be applied to a case that contains conditions in access control.

Suppose there are several basic capacities in an enterprise which are Manager, Clerk and Visitor. And the corresponding Permissions they have are illustrated in the Table below.

Table 4.1: Case Study Examples

Roles	Permissions	Conditions
Manager	Gate	
	Approve access of Database	
	Database	
	Wireless Network	In the building
	Elevator	Not stop at 2^{nd} floor
	Web Site	
Clerk	Gate	Weekday
	Database	Approval from the manager
	Wireless Network	In the building
	Elevator	Not stop at 2^{nd} floor
	Web Site	
Visitor	Elevator	Not stop at 2^{nd} floor
	Web Site	

There are five resources can be acquired which are Gate, Database, Wireless Network, Elevator and Web Site. Some of them must be accessed under certain conditions. This case contains those authorization ways that may present in actual applications. Those are the principles of condition-aware access control we need to pay attention.

- Who can access what resources.

- Who with what attributes can access what resources.
- With what attributes the resources can be accessed.

We try to model these authorizations systematic. Next, we apply RBAC, LRBAC, GRBAC to model this scenario one by one. Then we use our model and illustrate the differences between the previous works.

4.1 Case Study on RBAC

RBAC can only describe the authorization by user identity. In this case, RBAC can't distinguish the authorization which has conditions. The PA may be applied like the table below.

Table 4.2: Case Study of RBAC - Permission Assignment

Roles	Permissions
Manager	Gate, Approve access of Database, Database, Wireless Network, Elevator, Web Site
Clerk	Gate, Database, Wireless Network, Elevator, Web Site
Visitor	Elevator, Web Site

In this table, all permissions are directly assigned to those roles. Without considering the conditions, the security requirement may not be satisfied. The following figure shows the Role inheritance.

4.2 Case Study on LRBAC

Basically, LRBAC focuses on the user location. It can take user location as a constraint on User Assignment, but it didn't mention about other conditions. So the PA may appear like the table below.

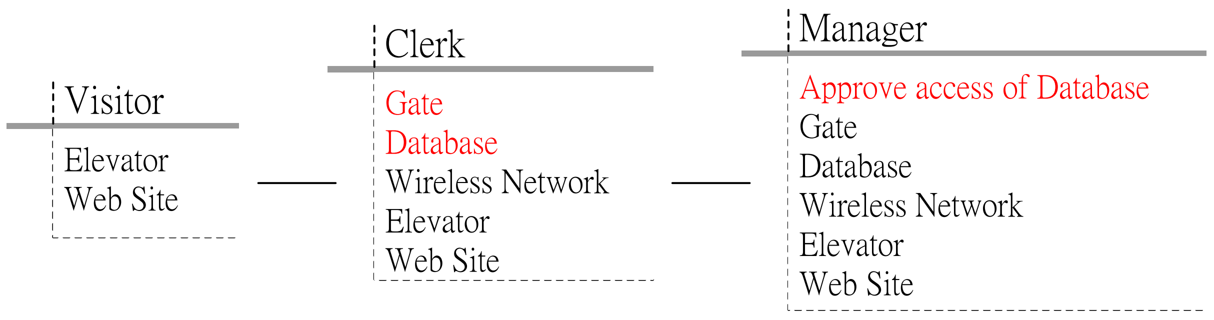

Figure 4.1: Case Study of RBAC - Hirarchical Roles

Table 4.3: Case Study of RBAC - Permission Assignment

Roles	Permissions
Manager	Gate, Approve access of Database, Database, Elevator, Web Site
Manager(in the building)	Wireless Network
Clerk	Gate, Database, Elevator, Web Site
Clerk(in the building)	Wireless Network
Visitor	Elevator, Web Site

Manager and Clerk can access the wireless network only in the building. LRBAC can describe these roles which interact with locations, but it still can't describe those conditions other than locations like "Weekday". However, LRBAC can be extended to include other conditions and takes them as a constraint on UA just like locations. Suppose we call it LRBAC', LRBAC' can model the conditions in this case. The PA becomes as below.

In order to describe every conditions, LRBAC' needs ten roles. LRBAC' needs to create role for every independent condition. With the growing of number of condition, LRBAC' may use a lot of roles and some of them are essentially redundant. Management overhead may increase in this case.

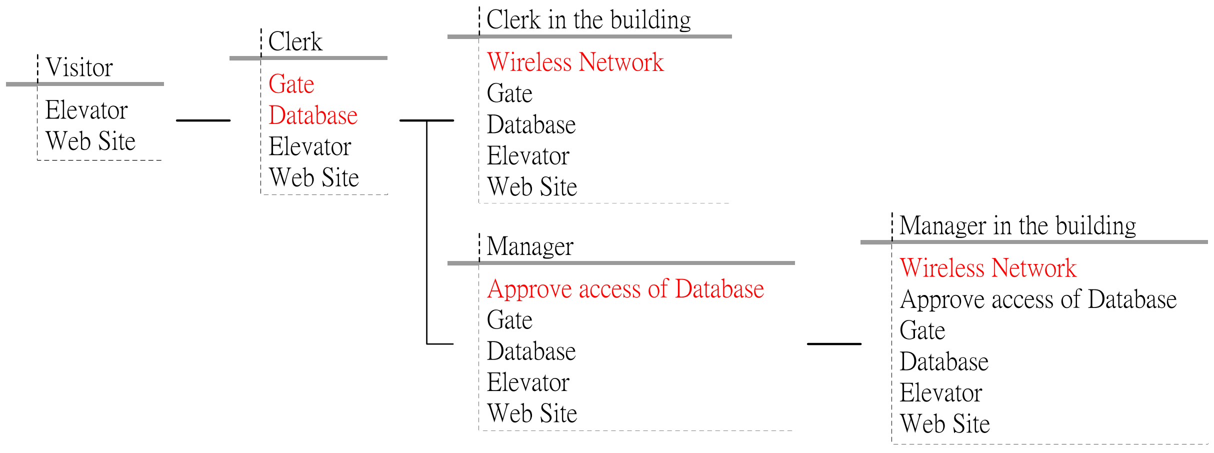

Figure 4.2: Case Study of LRBAC - Hirarchical Roles

Table 4.4: Case Study of LRBAC' - Permission Assignment

Roles	Permissions
Manager	Gate, Approve access of Database, Database, Web Site
Manager(in the building)	Wireless Network
Manager(Not stop at 2^{nd} floor)	Elevator
Clerk	Web Site
Clerk(in the building)	Wireless Network
Clerk(in Weekday)	Gate
Clerk(with Approval from Manager)	Database
Clerk(Not stop at 2^{nd} floor)	Elevator
Visitor	Web Site
Visitor(Not stop at 2^{nd} floor)	Elevator

4.3 Case Study on GRBAC

GRBAC can support full conditions in this scenario. It just implements the basic roles and takes all conditions as constraints on PA. The PA is illustrated on the table below.

Manager is pre-assigned those permissions which don't interact with conditions like "Gate" and "Database". User can directly acquire these permissions after their identity being verified as Manager. And for those permissions which interact with conditions, GRBAC frequently verifies their status and decides if the role can have these permissions. So GRBAC just needs three roles

Table 4.5: Case Study of GRBAC - Permission Assignment

Roles	Permissions	Conditions*
Manager	Gate, Approve access of Database, Database, Web Site	
	Wireless Network	In the building
	Elevator	Not stop at 2nd floor
Clerk	Web Site	
	Gate	Weekday
	Database	Approval from Manager
	Wireless Network	In the building
	Elevator	Not stop at 2nd floor
Visitor	Web Site	
	Elevator	Not stop at 2nd floor

but there are four conditions need to be verified frequently.

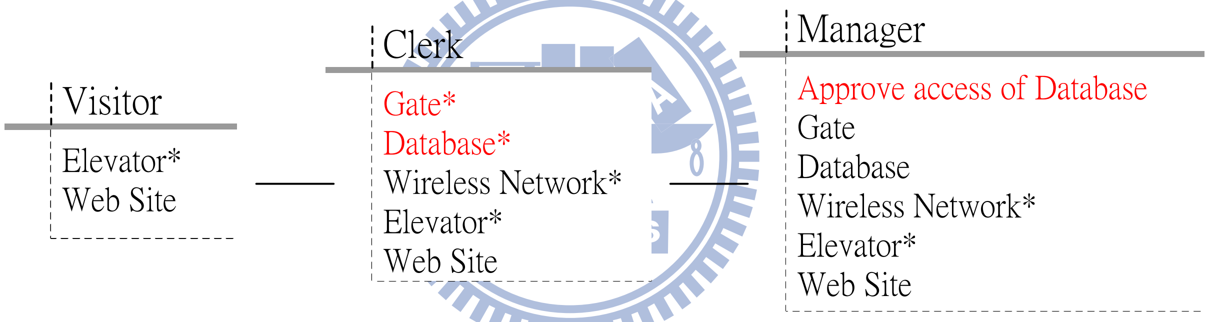

Figure 4.3: Case Study of GRBAC - Hirarchical Roles

4.4 Case Study on Our Model

Finally, we apply our model. First, we separate the mutable and immutable conditions. Considering the access operation time period, Conditions "Approval from Manager" and "In the building" may vary during the access operation "Database" and "Wireless Network" respectively. But conditions "Not stop at 2^{nd} floor" and "Weekday" don't change during access to "Gate" and "Elevator". So we create roles for those immutable conditions. The PA appears like the table below.

Table 4.6: Case Study of Our Model - Permission Assignment

Roles	Permissions	Conditions
Not stop at 2nd floor	Elevator	
Manager	Gate, Approve access of Database, Database, Elevator, Web Site	
	Wireless Network	In the building
Clerk	Web Site	
	Database	Approval from Manager
	Wireless Network	In the building
Clerk(in Weekday)	Gate	
Visitor	Web Site	

In this case, our model needs five roles to illustrate the authorization. And there are two conditions need to be verified frequently. Also, it can satisfies all authorization scene in this case. Next chapter, we give a comparison between our model and other proposals. After the analysis, we point out how much benefit we can obtain by applying our access control model.

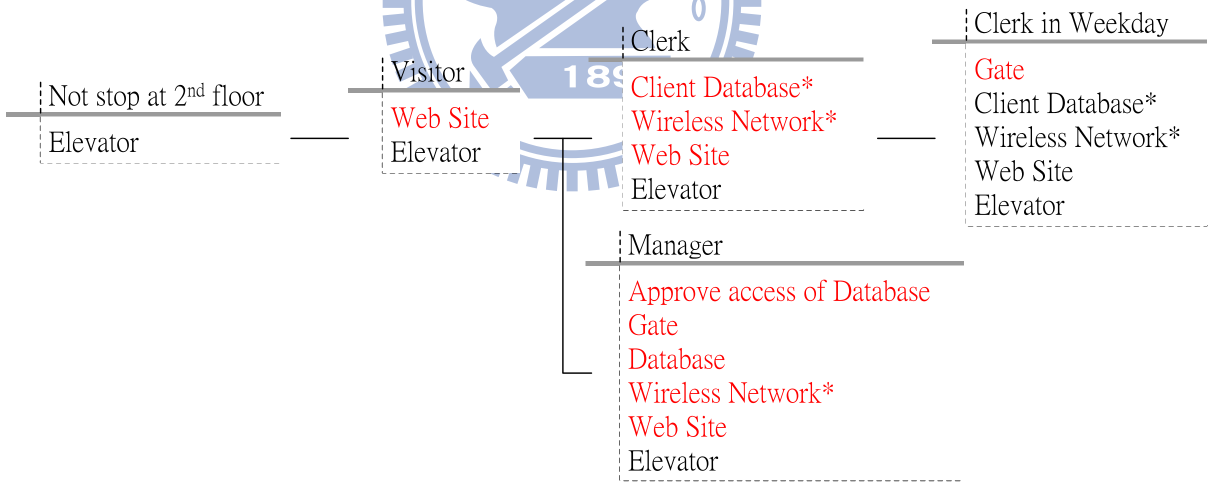

Figure 4.4: Case Study of Our Model - Hirarchical Roles

Chapter 5

Analysis on Overheads

The original authorization treats every permission as an independent entry. Therefore, with the increase of permissions, administrator needs to deal with a large amount of authorization. Without a good arrangement, it's impractical to assign every permission to every user separately. In order to reduce the management overheads, RBAC introduces the concept of role. Basically, in an organization, the combination of permissions may be only a few types depending on the capacity of user. Applying role in access control can make significant reduction in authorization entries, and then reduce the management overheads. So the number of authorization entries or say the number of roles is one of the factor that effect management overheads.

Another factor is the frequency of verifying conditions and roles. Less time we need to verify conditions, less overheads both on the system and management. Basically, the frequency of verifying conditions is dependent with the type of condition. But we can control the number of conditions we verified. Treating the immutable conditions as roles can reduce the frequency of verifying roles. User can acquire those permissions which are mutable with conditions directly by their roles and doesn't have to wait for the condition verification. And the condition verification only was made when those permissions associated with mutable conditions are being request.

5.1 Management Overheads

Comparing those proposals applied on the case study illustrated in the previous chapter, we can obtain the difference of capability as the following table.

Table 5.1: Analysis on Management Overheads

Authorization Theme	RBAC	LRBAC	GRBAC	Our Model
identity	○	○	○	○
context	×	×	×	○
identity+location	×	○	○	○
identy+context	×	△	○	○
number of roles in the case	3*	5**/10	3	5

*RBAC constructs 3 roles in the scenario but can't model the authorization with context.
**LRBAC constructs 5 roles to model the authorization with location, but it needs 10 roles to model the authorization with context.

To support the authorization with context information, RBAC is insufficient and LRBAC needs to be modified. We call the modified LRBAC as LRBAC' and comparing with GRBAC and our model.

Because of taking all conditions constraints as roles, LRBAC' construct way more of roles. This increases the complexity of roles hierarchy as Figure 5.1.

Instead of constructing all condition constraints as roles, our model takes mutable conditions as constraints on Permission Assignment and reduces a great number of roles. Similarly, GRBAC makes all condition to restrain permissions assigning to roles. GRBAC can have less number of roles than our model. But since all conditions need to be verified frequently, it makes user hard to ascertain their permissions by their role and increases the overhead on condition verification. We will discuss it in the next section.

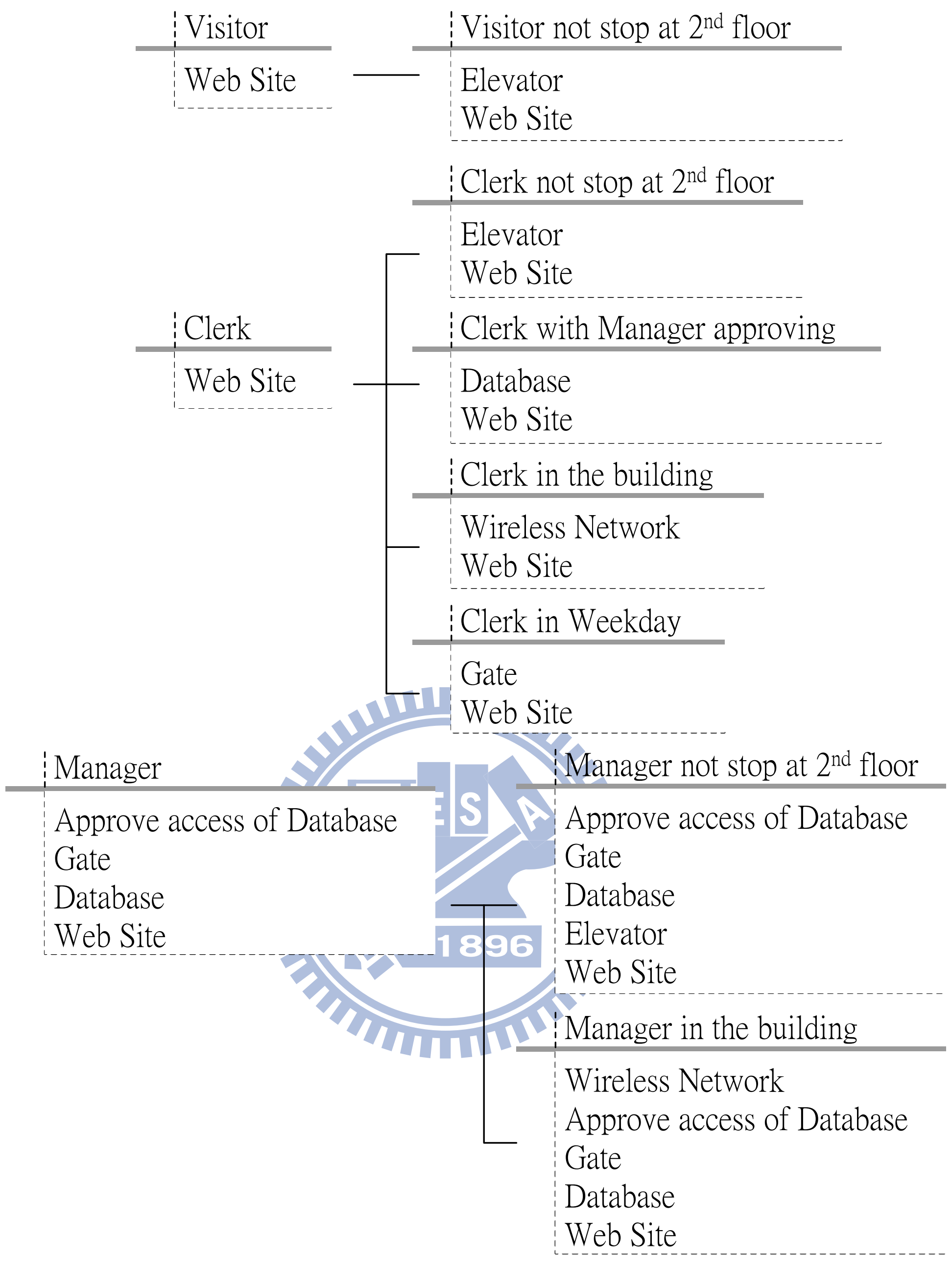

Figure 5.1: Case Study on LRBAC'

5.2 Condition Verification Overheads

In the case study, there are four kinds of condition constraints. Suppose the varying period of each condition denote as τ_i and the period of corresponding access if α_i. If $\tau_i \gg \alpha_i$, which means this kind of condition seldom change during the access. Then we set the corresponding condition as immutable condition, otherwise, as a mutable condition. Condition "Not stop at

2^{nd} floor" and "Weekday" are immutable in the case study. So we construct Role "Clerk(in Weekday)" etc., leaving Condition "In the building" and "Approval from Manager" as mutable conditions. User push the information of "Who they are", "if it is Weekday" and "which floor he rides the elevator to" to the access decision point with access request. This information doesn't have to be verified during once access. So we verify two less conditions than GRBAC.

Suppose the access frequency of variation of mutable conditions be fc_i. The frequency of verification must greater or equal than the frequency of variation and be $fv_i \geqslant fc_i$. As a result, the verification time with an access should be $fv_i \times \tau_i$ for a related condition. Suppose a permission is related two n conditions, the total verification time within an access should be

$$\sum_{i=1}^{n} fv_i \times \tau_i \tag{5.1}$$

Making the immutable conditions as roles and only verified with the request can reduce the overhead on condition verification. Also, users can know what they are allowed to do more directly by their roles.

Chapter 6

Conclusion and Future Work

In this research, we design a distinctive access control model to relax management overheads. Our model is based on Role-based access control and incorporates the concept of condition. Condition includes context information and user interaction and can be the constraint on authorization. Mainly, we introduced three ways to make access control decision. First, user acquire role to get permission by its identity and immutable condition. Second, permission which assigned to role varies depending on mutable condition. Third, user gets permission only by the condition they has. By separating mutable from immutable condition, user pushes their identity and immutable condition to the access control decision point to grant permission intuitionally. Then the access control decision point frequently updates mutable condition to reduce the number of roles. Then we illustrate a case study on previous proposal, our model establishes less roles and the frequency of condition verifying is acceptable. We also present our model by XACML to make it easier to communicate around the cyberspace and anticipate it to adapt in many applications. Overall, our model provides an access control mechanism that is direct and simple. Though our model tends to be general and the type of condition we mentioned is not complete enough, it still can be improve to cater more specific application and more complex condition.

References

[1] R. Sandhu and P. Samarati, "Access control: principle and practice," *Communications Magazine, IEEE*, vol. 32, no. 9, pp. 40--48, Sep 1994.

[2] D. E. Denning and P. F. MacDoran, "Location-based authentication: grounding cyberspace for better security," pp. 167--174, 1998.

[3] A. Prayogi, J. Park, and E. Hwang, "Selective role assignment on dynamic location-based access control," *Convergence Information Technology, 2007. International Conference on*, pp. 2136--2135, Nov. 2007.

[4] F. Hansen and V. Oleshchuk, "Spatial role-based access control model for wireless networks," *Vehicular Technology Conference, 2003. VTC 2003-Fall. 2003 IEEE 58th*, vol. 3, pp. 2093--2097 Vol.3, Oct. 2003.

[5] R. Sandhu, E. Coyne, H. Feinstein, and C. Youman, "Role-based access control models," *Computer*, vol. 29, no. 2, pp. 38--47, Feb 1996.

[6] R. Sandhu, D. Ferraiolo, and R. Kuhn, "The nist model for role-based access control: towards a unified standard," pp. 47--63, 2000.

[7] "Nist - role based access control (rbac) and role based security," Tech. Rep. [Online]. Available: http://csrc.nist.gov/rbac/

[8] D. F. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramouli, "Proposed nist standard for role-based access control," *ACM Trans. Inf. Syst. Secur.*, vol. 4, no. 3, pp. 224--274, 2001.

[9] M. J. Covington, W. Long, S. Srinivasan, A. K. Dev, M. Ahamad, and G. D. Abowd, "Securing context-aware applications using environment roles," pp. 10--20, 2001.

[10] M. Moyer and M. Abamad, "Generalized role-based access control," *Distributed Computing Systems, 2001. 21st International Conference on.*, pp. 391--398, Apr 2001.

[11] M. Covington, M. Moyer, and M. Ahamad, "Generalized role-based access control for securing future applications," 2000.

[12] I. Ray, M. Kumar, and L. Yu, "Lrbac: A location-aware role-based access control model," *Lecture Notes in Computer Science*, vol. 4332, p. 147, 2006.

[13] I. Ray and M. Kumar, "Towards a location-based mandatory access control model," *Computers & Security*, vol. 25, no. 1, pp. 36--44, Feb 2006.

[14] D. Kulkarni and A. Tripathi, "Context-aware role-based access control in pervasive computing systems," pp. 113--122, 2008.

[15] R. J. Hulsebosch, A. H. Salden, M. S. Bargh, P. W. G. Ebben, and J. Reitsma, "Context sensitive access control," pp. 111--119, 2005.

[16] A.-T. Ferreres, B. Alvarez, and A. Garnacho, "Guaranteeing the authenticity of location information," *Pervasive Computing, IEEE*, vol. 7, no. 3, pp. 72--80, July-Sept. 2008.

[17] A. Sayed, A. Tarighat, and N. Khajehnouri, "Network-based wireless location: challenges faced in developing techniques for accurate wireless location information," *Signal Processing Magazine, IEEE*, vol. 22, no. 4, pp. 24--40, July 2005.

[18] C. A. Ardagna, M. Cremonini, E. Damiani, S. D. C. di Vimercati, and P. Samarati, "Supporting location-based conditions in access control policies," pp. 212--222, 2006.

[19] C. Ardagna, M. Cremonini, S. D. C. di Vimercati, and P. Samarati, "Privacy-enhanced location-based access control," 2007.

[20] "Oasis extensible access control markup language (xacml) tc," Tech. Rep. [Online]. Available: http://sunxacml.sourceforge.net/guide.html

[21] "Extensible access control markup language (xacml)," Tech. Rep. [Online]. Available: http://xml.coverpages.org/xacml.html

[22] A. Anderson, "Core and hierarchical role based access control (rbac) profile of xacml v2.0," *OASIS Standard*, 2005.