Technical Report

UCAM-CL-TR-558
ISSN 1476-2986

Number 558

A role and context based security model

Yolanta Beresnevichiene

January 2003

This technical report is based on a dissertation submitted June 2000 by the author for the degree of Doctor of Philosophy to the University of Cambridge, Wolfson College.

Series editor: Markus Kuhn

ISSN 1476-2986

Abstract

Security requirements approached at the enterprise level initiate the need for models that capture the organisational and distributed aspects of information usage. Such models have to express organisation-specific security policies and internal controls aiming to protect information against unauthorised access and modification, and against usage of information for unintended purposes. This dissertation describes a systematic approach to modelling the security requirements from the perspective of job functions and tasks performed in an organisation. It deals with the design, analysis, and management of security abstractions and mechanisms in a unified framework.

The basis of access control policy in this framework is formulated around a semantic construct of a role. Roles are granted permissions according to the job functions that exist in an organisation, and then users are assigned to roles on basis of their specific job responsibilities. In order to ensure that permissions included in the roles are used by users only for purposes corresponding to the organisation's present business needs, a novel approach of "active" context-based access control is proposed. The usage of role permissions in this approach is controlled according to the emerging context associated with progress of various tasks in the organisation.

The work explores formally the security properties of the established model, in particular, support for separation of duty and least privilege principles that are important requirements in many commercial systems. Results have implications for understanding different variations of separation of duty policy that are currently used in role-based access control.

Finally, a design architecture of the defined security model is presented detailing the components and processing phases required for successful application of the model to distributed computer environments. The model provides opportunities for the implementers, based on application requirements, to choose between several alternative design approaches.

Contents

Chapter 1

Introduction

1.1 Secure System

Information is an asset for every organisation. The constantly increasing impact of computer systems on the functioning of organisations results in concerns about threats to the information usage. Unfortunately, some basic characteristics of computer systems, that provide overall availability of data and system services, imply risks to information. The inability of computer system to protect the confidentiality and integrity of information may cause serious financial and legal problems in an organisation.

These concerns have all contributed to security, approached from an information-oriented perspective, becoming an important issue in a computer system. The primarily objectives of the security model presented in this work are expressed in terms of protecting information against the unauthorised access and modification, and against the usage of information for unintended purposes.

1.2 Representation of Reality

In order to be helpful and not misleading, security models must be developed not just from the formed preconceptions about the security issues in computer systems, but by examining the security requirements arising from the context of some operational and threat environment. The concepts and principles associated with developing a security model must reflect the characteristics of this environment.

Many existing security models focus on expressing the requirements that must hold within the components internal to a computer system. This approach has contributed to the development of effective mechanisms and tools for implementing internal security specifications within computers. However, the view taken from an organisation that owns a computer system expresses different needs and trends. The emphasis here is on policy-oriented as opposed to mechanism-oriented abstractions and facilities. The emerging security concerns require to represent various authorisation and control policies related to job functions and business activities that operate within a distributed computing environment.

1.3 Research Motivation

The traditional security models have been formulated at too low level of abstraction to be useful for modelling the security requirements that express the organisational and distributed aspects of information usage. In particular, these models are primarily concerned with security aspects of internal data components and related rules of low level operations, and thus lack the constructs necessary to model the job function-related controls and activity-related authorisations.

Although the work towards formulating the conceptual foundations of higher-abstraction level security models has advanced with recent research efforts, this dissertation was motivated by the need to deal with many related security issues of an organisational level in a unified framework.

1.4 Research Statement

This work proposes a security model for access control and authorisation management. It captures the security requirements that arise from an organisational context where responsibilities are divided according to roles and activities are undertaken within tasks. The aim is to produce a clear and versatile model that provides the abstractions and mechanisms to (a) model and enforce access control from the perspective of roles, (b) actively manage authorisations in the context of business activities, and (c) support the issues of distributed environment.

1.5 Outline of Dissertation

Chapter 2 considers organisational requirements for security models. It gives motivations for the role-based modelling of security and argues the necessity for contextual interpretation during the enforcement of security.

Chapter 3 outlines the basic framework of our access control model. It presents essential concepts, definitions of the main entities, and formal specifications of mapping functions and access granting rules.

Chapter 4 analyses security constraints necessary to enforce organisational security policies of separation of duty and Chinese Walls within the framework presented in chapter 3.

Chapter 5 introduces context-based authorisations as means to control the activation of role-related rights from the perspective of progressing tasks. The mechanisms are described that make it possible to interwork with transactional workflows, and that provide for the dynamic enforcement of access control.

Chapter 6 proposes a design architecture for application of the defined security model into computer systems. It also gives detailed description of the entities involved in each of the three processing phases associated with the design.

Chapter 7 looks at authentication, as one of the fundamental concerns when applying the proposed architecture to distributed computing environments.

Chapter 8 summarises the research results and suggests further work.

Chapter 2

Security Models and Organisation

2.1 Introduction

This chapter discusses requirements for security models. In particular, we concentrate upon security concerns of commercial organisations, that are characterised by a dynamic environment of activities being accomplished according to the task fulfilment responsibilities and the actual business processing. In the following sections we derive the set of requirements arising from an organisational level, and motivate the need for role-based control and contextual interpretation within a security model.

2.2 Security Goals

Any modelling of the computer system's security must be towards reaching the security goals. The main objectives of security concentrate upon protecting information from unauthorised disclosure and preventing unauthorised actions and improper modification of information.

Therefore, the essence of a security model must be to ensure that system users can only gain access to those system resources, including information and programs, for which they have proper authorisations[1]. The basis for these authorisations is mainly derived not from the ability and trust placed upon a user, but from a legitimate need to access the resources. The needs are formulated in advance by analysing and modelling requirements, that originate in a realistic computing and organisational environment.

2.3 Modelling of Security

It is widely recognised that security requirements can be viewed at different levels of abstraction. LaPadula and Williams [LW 91] proposed a layered taxonomy of stages where the security requirements at higher levels can be refined and elaborated to lower levels. Starting with the highest level, these include:

1. *Trust Objectives:* the basic organisational security objectives to be achieved by a system.

[1]Providing user accountability for their authorised actions is another important goal.

2. *External-Interface Requirements:* the system's interface to the environment, in terms of the security requirements.

3. *Internal Requirements:* the requirements that must hold within the components internal to a system.

4. *Rules of Operation:* the rules that explain how internal requirements are enforced.

5. *Functional Design:* a functional description of the behaviour of system components.

The security requirements of a system at stages 1 and 2 are at much higher level of abstraction than those at stages 3, 4, and 5. The higher stages specify *what* needs to be done, and these get refined into detailed executable specifications that deal with *how* things are to be done.

Given these stages of elaboration, one can formulate security models for each of these stages, as well as classify existing models as to where they belong. Therefore, at the highest level there are security models that capture elements of organisational policies and requirements that pertain to security. These requirements are, through interfaces, refined and captured by tools and mechanisms in computer-oriented implementation models.

Previous or traditional development has been aimed at specifying and implementing internal mechanisms based on a computer-oriented policy. Traditional approaches are of two categories, mandatory and discretionary, depending on the type of policy they apply. Mandatory approach restricts the access of users to the information on the basis of security levels and classification of subjects and objects to these levels. In a discretionary approach access is based on a user's identity and users are permitted to allow or disallow other users access to objects under their control.

Although both discretionary and mandatory models have been successfully used, there is a need for richer and more flexible security models. This need is motivated by different requirements arising from the organisational level, namely those placed upon the specification and administration of access rights, and the means for expressing versatile security policies.

2.3.1 Access Rights

The common characteristic of traditional approaches is that they operate using simple access modes such as read, write, execute, that control access at the operating system level. At the application level, however, one would like to see access rights which relate to richer and more complex operations. On this level information is manipulated with transactions such as issue the invoice, authorise payment, debit or credit an account. These transactions embody access rights specified on a functional basis.

2.3.2 Ease of Administration

The administration of access rights in computer-oriented models is performed at the level of individual subjects and objects. Administrators distribute and review user access rights to individual objects. Instead of simply assigning or removing a user to or from a functional position in an organisation, an administrator has to access low level system objects and change access right to them. This often creates administrative confusion and errors.

2.3.3 Expressing Policies

Traditional approaches are oriented towards specific environments that do not support a larger variety of security policies. Mandatory policies are mostly useful for rigid requirements, like those of the military environment. Discretionary policies, on the other hand, rely on co-operation of users, arising from requirements of an academic environment. As noted by Clark and Wilson [CW 87], neither approach can effectively express many practical requirements, especially on provision and maintenance of integrity, arising in commercial organisations.

2.4 Organisational Requirements

This work attempts to bridge the gap between the internal requirements and higher stages of elaboration. It is intended to model organisation-specific security requirements. Usually such a security model is context dependent. An appropriate security model can be designed only when it is known for what sort of an organisation, for what structure and policies it will be used. However, the aim is not to present a model which is applicable only to the specific type of environments, but one which is general enough to capture common aspects of security in many organisations and also is flexible enough ‘to incorporate unique requirements of separate organisations.

The basis for organisational security models is derived from the structure of an organisation and the form of activities it undertakes, as well as from an explicitly or implicitly expressed security policy.

2.4.1 Structure and Activities

Organisational theory [TH 90] emphasises features common to all organisations: they all operate by division of labour, or more generally, division of complexity. The primary activity of an organisation, such as manufacturing of cars, clothes, etc., being a large undertaking, is divided into smaller, more manageable, and co-ordinated **tasks**. The division of activity produces many functional tasks that are performed in an organisation on regular basis.

In any organisation larger than one person there is also division of work between individuals. This division produces **roles** which individuals occupy. Each role defines the responsibilities of the role member, and functions to be performed.

Role and task analysis of an organisation would therefore permit the expression of functional requirements and dependencies, that form the basis for analysing possible conflicts in a security model.

2.4.2 Security Policies

In addition, each organisation formulates its security policy to state acceptable and unacceptable situations and values, and to express safeguards that ensure which authorised individuals can access the specific information. It is commonly agreed [DMc 89] that these elements of policy must be reflected in the design of an appropriate security model.

Although security policies differ in every organisation, there exists a number of fundamental security principles, particularly applicable within commercial security systems:

- **Least Privilege**: according Salzer and Schroeder [Sal 75] every program and every user of the system should operate using the least set of privileges necessary to complete the job;
- **Separation of Duty**: two or more different people should be responsible for the completion of a task;
- **Chinese Wall**: people are only allowed access to information which is not held to conflict with any other information that they already possess.

The intent of the first principle is to limit user's access rights only to those that are required to perform a particular function. The support of this principle within a computer system reduces the number of privileges given to one user to a minimum, so that unintentional, unwanted, or improper use of access rights are less likely to occur.

The purpose of the second principle is to prevent fraud by separating a task into subparts and spreading the responsibility of each subpart over multiple people. Salzer and Schroeder [Sal 75] credited R. Needham for making the following observation: a protection mechanism that requires two keys to unlock is more robust and flexible than one that requires only a single key. Committing a fraudulent act then requires the involvement of more than one individual, making no single accident, deception, or breach of trust sufficient to compromise the system's security.

The Chinese Wall policy can be most easily visualised as the code of practice that must be followed by a market analyst working for a financial institution providing corporate business services. Such an analyst must uphold the confidentiality of the information provided to him by company's clients; this means that he cannot advise corporations where he has insider knowledge about a competitor. However, the analyst is free to advise corporations that are not held in competition with each other.

A security model must also provide means for expressing seniority in an organisation, mechanisms to allow a user to delegate a role or privilege to another user in case of absence or election (a deputy member), and methods to enforce various prerequisite conditions such as that of the duty officer, when a position must be occupied by one person at a time, different from the previous one.

2.5 An Alternative - Role-Based Controls

In the recent study of access control requirements in organisations conducted by the National Institute of Standards and Technology (NIST) [FGL 93], it has been found that access control decisions were based on "the roles individual users take on as part of an organisation". For example, the roles an individual associated with a hospital can assume include among others doctor, nurse, and a clinician. Roles in a bank include teller, loan officer, and an accountant.

When roles are introduced at the application level to control access to the application data, they offer an excellent opportunity to realise benefits in securing organisation's information assets, similar to the benefits of employing databases instead of files as the data storage system. Roles consolidate scattered access rights into a unified service which can be better managed while providing the flexibility and customisation required by individual applications.

Over the past years roles and role-based access control (RBAC) has been used in a variety of forms for computer systems security. Earlier work presented the proposals for incorporating roles into the existing security mechanisms, such as usage of mandatory controls and type enforcement [Thom 91], extensions to SQL security system and access control lists (ACLs) [Bal 90], and creation of a subset of trusted systems [Ste 92]. Introduction of roles at access control level draw attention to many security issues not covered by the traditional mechanisms and as such role-based access control has been separated into a policy-neutral alternative to classical discretionary and mandatory access controls.

Recently the definitions of basic concepts and main features involved in role-based access control were described by Ferraiolo *et al* [FKC 95, FK 92] and Sandhu *et al* [SCFY 96]. RBAC regulates the access of users to information and system resources on the basis of the particular function a user is allowed to perform in an organisation. Instead of specifying access rights for each individual user, access authorisations to objects, called permissions, are associated with roles. Roles are then allocated to users according to the current functionality requirements. Since the variety of roles is relatively persistent with respect to user turn-over and function re-assignment, RBAC provides a powerful mechanism for reducing the complexity, cost, and potential for error when administering access rights in a computer system. These advantages of RBAC have been extensively analysed by Gligor and Lorenz [GL 95] and many others.

Various new proposals have emerged targeting specific issues in role-based access control. In the next section we give a short overview of the previous and current work on role-based access control. We then proceed to indicate the emerging areas that we think are particularly in need for further consideration.

2.6 Related Work

Demurjian *et al* [DTHD 96, DTH 94, DTH 92] first proposed to implement role-based security in an object-oriented data model. Their approach was to exploit the object-oriented paradigm to organise user-roles and their permissions. A framework was presented for organising users into a user-role definition hierarchy.

Several proposals have concentrated on a formal approach to describe role configuration. Giuri [GI 95] presented an alternative definition of a role as a named set of protection domains (NSPD). A formal model for managing NSPDs and a formal specification of its semantics was introduced that takes into account the capability to support security constraints. Nyanchama and Osborn [NyOs 94] described a formal semantics for expressing role relationships. The role graph model with its operator semantics and algorithms based on graph theory was proposed to organise roles. The work built an extensive framework for organising and managing many different role relationships.

The initiative started by NIST resulted not only in describing the main features of role-based access control [FKC 95], but also in isolating several fundamental issues of RBAC.

Because roles within an organisation typically have overlapping permissions, RBAC has to include some means to establish and maintain role hierarchies, where a role can include the permissions of other roles. Role hierarchies have been extensively discussed by Jansen [Jan 98] and Sandhu [SA 98]. Another fundamental aspect of RBAC is specification of security constraints required to enforce a particular organisational security policy. The importance of constraints in RBAC has been recognised both by Sandhu *et al* in [SCFY 96] and Ferraiolo *et al* in [FKC 95], but beyond that they have not received much attention in the research literature.

2.7 Emerging Areas

The aim in this work is to present a security model that includes the basic elements of RBAC, and that in addition extends RBAC with novel features that have not been addressed previously. Two main emerging areas are enforcement of security constraints and support for an "active" approach to access control.

2.7.1 Security Constraints

Authorisation constraints are an important aspect of access control and are a powerful mechanism for laying out higher level organisational policy. With the help of constraints the designers can lay out a broad scope of what is acceptable, and make it across administration domains. Unfortunately, the analysis and enforcement of constraints within role-based security systems has been only preliminary and tentative.

In describing our security model we particularly focus on specification and enforcement of constraints. The objective is to present a constraint specification framework that provides for the support of the fundamental security principles indicated in section 2.4.2. We will treat constraints as invariants that should hold at all times, and specify them formally using first order predicate logic.

2.7.2 "Active" Control

While discussing the major risks for information systems in organisations, Holbein and Teufel [HT 95] noted that an extended support should be provided for ensuring that information in a computer system is accessed and modified only for intended purposes. The misuse of allocated access rights by authorised users for purposes that are neither intended nor acceptable by an owner of information (in our case, an organisation) represents a serious breach of security. Several studies during the last years [CSRB 92] also confirm that the major threats to information are caused not by outside attackers, but by internal persons in the organisation who are authorised and in confidence.

The need to access the information and programs depends on the current business activities in an organisation which imply that a particular task must be fulfilled, e.g. an order must be verified, or shipment of an ordered product must be done. Therefore, activation of allocated permissions by authorised users must correspond to the organisation's current business transactions. This implies, that permissions should not persist beyond the time that they are required for performance of a task. In order to support the timely revocation of permissions a security model has to provide mechanisms that recognise the overall context in which security requests arise and take an active part in the management of security as it relates to the progress and emerging context within tasks.

Our intention in this work is to show that RBAC framework can be extended further so that in result the security model provides for the support of “active” access control.

2.8 Conclusions

In this chapter we have reasoned that there exists a need to model security at the organisational level. The security requirements at this level arise from dependencies between job positions and task fulfilment responsibilities. Such requirements cannot be expressed just by computer-level operations and mechanisms implemented in traditional security models. An alternative, more flexible approach was considered where the central view of security moves from the concrete subjects and objects to higher abstractions refereed by roles and tasks. With the help of roles access needs in an organisation can be easily managed. In addition, contextual interpretation of progressing tasks can be used to prevent unintended usage of access rights embodied in roles.

Chapter 3

Basic Framework

3.1 Introduction

In the previous chapter an alternative approach to access control was highlighted, which is flexible, policy independent, and closely reflects the versatile environments of organisations. A role was introduced as the basic construct of this approach.

A role is usually seen as a job position or a named job function performed within an organisation that embodies authority, responsibility, or competency [FK 92, SCFY 96]. In a security model, a role is used to represent job related access rights which can then be authorised to users as a single unit. A role must exist as an entity separate from the specific role holder, and be equipped with sufficient functionality to enable an authorised user to achieve its associated duty requirements.

In this chapter we give the precise definition of a role and of other entities that form the basis of a Role and Context Based Security (RCBS) model that is presented in this work. We also introduce the formal notation which will be used to formally specify rules and properties of the model throughout the rest of this work. The formalisms are described using the first-order predicate logic, the detailed overview of which can be found in [ML 93].

3.2 Basic Strategy

When designing an access control model, the first issues that must be addressed are the definitions of subjects and objects, and the types of access provided by the system.

3.2.1 Subjects and Objects

In respect to the RCBS model we define two types of subjects: the users of the system and the transactions that execute on behalf of those users. Users can access objects only by executing transactions on these objects.

A transaction here refers to the transformation procedure (a program, or a set of executable operations), which upon invocation manipulates data items or causes consumption of a system resource. The behaviour of the transaction and the type of data it can operate on are determined at design time. A transaction embodies different methods and granularity than simple access modes do, as can be seen in the following example. Tellers in

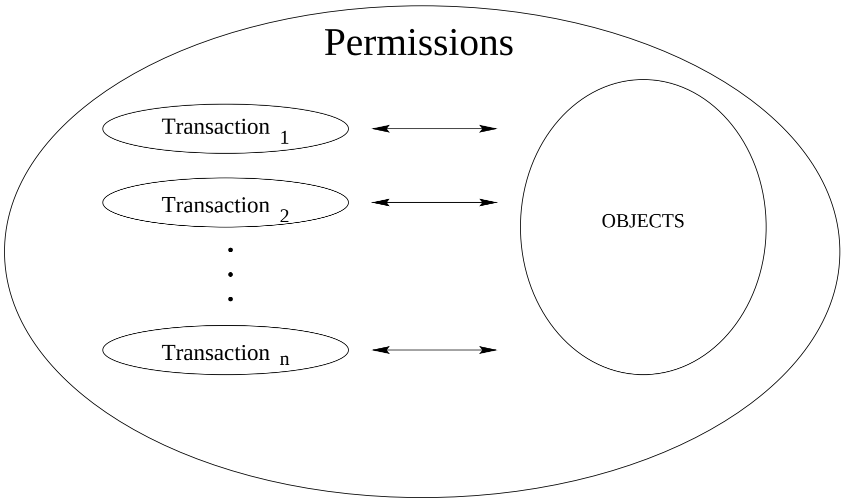

Figure 3.1: Permissions.

a bank are able to execute a savings deposit transaction, requiring read and write access modes to the specific fields within a savings file. An account supervisor is allowed to perform correction transactions, requiring the same read and write access modes to the same file as the teller's. The difference between these two transactions is in the whole process executed and in the values written to the transaction log file.

3.2.2 Permissions

The system protection in an access control model is realised in terms of *permissions*[1]. It is generally accepted that a permission describes an approval of a particular access right to an object or set of objects.

In presenting the RCBS model we are concerned with protecting computer system resources and data from unauthorised access and modification by *users* of this system. Since we have accepted that the only access rights the users have is to perform transactions, a permission in this work is viewed as the right to execute a particular transaction on a specific object from the defined set of objects (figure 3.1). For example, a permission may authorise the use of a spread-sheet package on specific files, or the execution of an "approve order" transaction for orders less than 1.000 pounds.

Definition 1 (Permission) *A permission* $\boldsymbol{p}$ *is a pair* $< \boldsymbol{tr}, \boldsymbol{objset} >$, *where* $\boldsymbol{tr}$ *is the transaction that operates on the given set of objects* $\boldsymbol{objset}$ *.*

Let P denote the universal set of permissions, Tr the universal set of transactions, and Obj the set of objects. We can define permission/transaction and permission/object associations with the following mapping functions:

$TrP(p) : P \rightarrow Tr$, gives the transaction associated with the given permission p,

[1]The term of a privilege is sometimes used instead of a permission.

$ObP(p) : P \rightarrow 2^{Obj}$, gives the set of objects associated with the given permission p.

In respect to the above definition permissions describe what a user is allowed to do, and do not define what a particular transaction is authorised to do. The issue of correctly processing transactions themselves is equivalent to proof of program correctness, while assurance that they access only the authorised data may be provided by application of internal security mechanisms [2]. These areas are outside the scope of this work.

3.3 Concept of a Role

A role is formed by grouping permissions according to the logical and functional requirements which this role must represent. Each role has the name which uniquely identifies this role in the system.

Definition 2 (Role) *A role* $\boldsymbol{r}$ *is a pair* $\boldsymbol{< rname, rpset >}$, *where* $\boldsymbol{rname}$ *is the role name, and* $\boldsymbol{rpset}$ *is the set of role permissions.*

Let R denote the universal set of roles. We express the mapping between roles and permissions with a function:

$PR(r) : R \rightarrow 2^P$, which gives the set of permissions for the given role r.

While allocating permissions to roles the adherence to *the principle of least privilege* should be ensured. This means that each role must be given no more permissions than are necessary for its functional requirements. Although the proposed model supports the implementation of this principle, it does not enforce the way this principle is applied. For that, the nature of each job in an organisation must be examined and the minimum set of permissions required for this job function determined. The role can then be restricted to permissions only from the specified domain.

3.3.1 User-Role Authorisations

The permissions associated with a role are administered as a single unit such that authorisation to access a role puts all the role's permissions at the disposal of an authorised user and thus confers access rights grouped in the role to the user. A role acts as a gateway to system permissions and accessible information, illustrated in figure 3.2.

Associated with each role is *a user authorisation list.* This list identifies the users who are authorised to access the role.

Definition 3 (User Authorisation List) *A user authorisation list* $\boldsymbol{UAL}$ *is of the form* $[\boldsymbol{u_1, u_2, ..., u_n}]$, *where* $\boldsymbol{u_i}$ *is the user identifier.*

The authorisation of access rights for an individual user is accomplished at two levels: either *explicitly* by assigning/revoking the user to/from an authorisation list of a role,

[2]Zakinthinos [Zak 93] presents a mechanism that implements the security principle of least privilege for user processes.

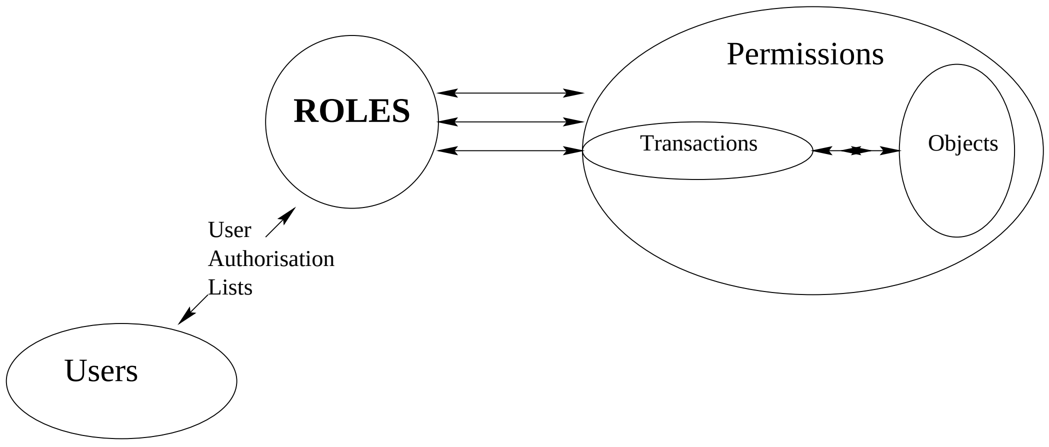

Figure 3.2: User-Role-Permission Mapping.

or *implicitly* by inclusion/removal of permissions in the role to which the user is already authorised.

Let U denote the set of user identifiers. Then a user authorisation list can also be defined as the mapping function between roles and users:

$UR(r) : R \rightarrow 2^U$, which enumerates the users associated with the given role r.

Although each role is required to have a user authorisation list associated with it, in some cases this list may be empty. (a) The permissions of a role may not be presently used in the computer system. The role with this set of permissions can still exist, but not be authorised to any user. (b) When hierarchies are defined for structuring of roles, the permissions of the role can be authorised to users indirectly through permission inheritance by other roles. A role will have an empty user authorisation list, but its permissions can be used by users authorised to senior roles[3].

3.3.2 Groups and Roles

In terms of user-role authorisation management, roles provide a superficial resemblance to the established concept of user **groups**, which is widely used for access control purposes, especially in association with access control lists (ACLs). It is, therefore, necessary to clearly state the major differences between these two concepts (figure 3.3).

1. The primary purpose for creating groups is to collect users according to their responsibilities. Only later permissions are associated with groups of users.

   In contrast, roles are created to collect unique sets of permissions, each being the set of permissions necessary to carry out some associated duty responsibilities. Users are then authorised to roles.

[3]The inheritance and indirect authorisation of permissions is discussed in more detail later in this chapter.

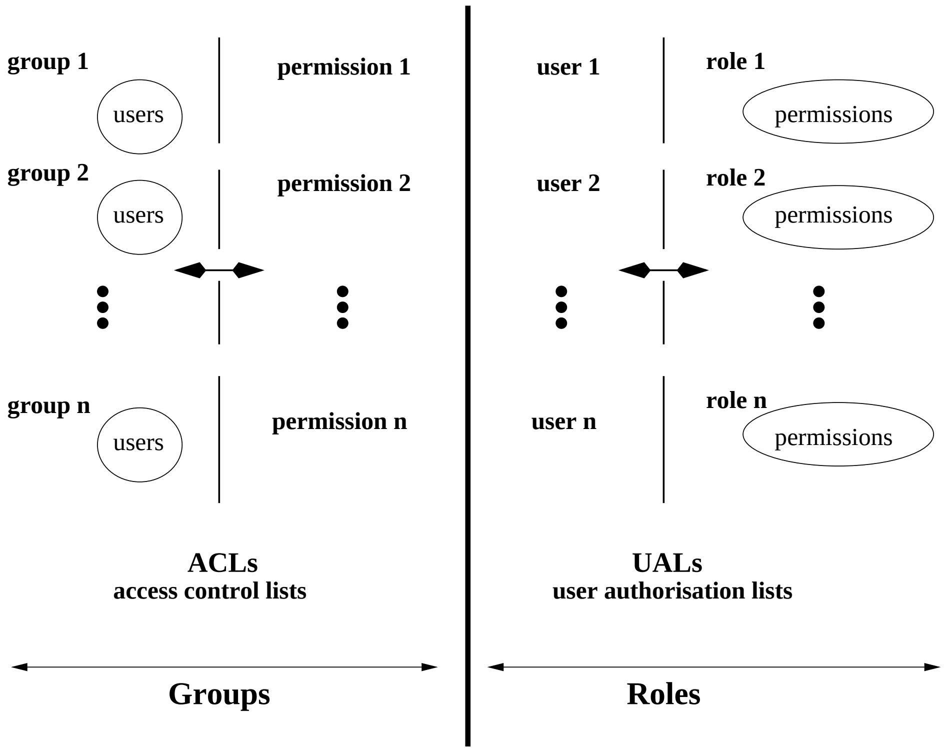

Figure 3.3: Groups and Roles.

2. A user, after being assigned to a group, is a member of this group at all times and in all circumstances. At every login session to the system a user is a member of the same previously assigned set of groups.

   When enforcing role-based access controls the issue of role activation inevitably arises. Even though a user is authorised to a role, he may not always need or be allowed to act in that role. In this case at each login session the user could be using a different set of roles [4].

An access control policy that uses groups and ACLs, and the one that is based on roles and UALs can both be seen as equivalent from the point that each can be used to describe the other. Given an access control policy of groups and ACLs an equivalent role-based policy can be constructed by authorising a user to a role if that user is a member of a group that maps to the same set of permissions as that of the role. On the other hand, a role-based access control policy can be transformed to groups and ACLs by making the user a member of the group and associating this group with the set of permissions that were assigned to a role to which the user was authorised. Barkley [Bar 97] presents a precise description of how such transformations can be accomplished.

The advantage of role-based control is that it eases the administration of permissions because of the flexibility with which roles can be configured and reconfigured. Given that permissions can be very fine-grained, roles offer means of managing then incrementally. Permissions can be easily revoked/assigned to roles, without affecting user authorisations

[4]Management of sessions and role activation will be discussed in chapter 6.

to roles. This approach offers a simplification of the complexity of permission management. Furthermore, in the next section we demonstrate that role relationships can be easily captured in hierarchical structures, allowing for management and analysis of permission distribution among roles.

3.4 Role Organisation

The exact configuration of roles addresses the issues of the specific field; role engineering is usually employed to establish the possible set of roles in an application. Although the content and structure of the final set of roles do not have direct implications on our model, it is important to identify what structurings and primitives are applicable to creating roles to be used by this model. In this section we present a generalised approach to different types of roles and discuss methods to be used for role organisation.

3.4.1 Identifying Role Types

In any organisation larger than one person the division of work produces many roles with differing responsibilities and job functions. Although the final set of roles depends on the structure of a specific organisation, many different roles can be divided into user-roles and functional roles.

An organisation consists of persons belonging to various administratively or physically divided staff groups, such as a department, a division, or a project team, that are referred as structural units. Various job positions and specific activities arising in structural units of an organisation are represented by *user-roles*. A user-role is associated with users that fill a job position or perform a specific activity, and permissions that describe this position or activity.

In addition to user-roles, general functions such as that of an engineer or of a programmer may exists perhaps common to several structural units. These are represented by *functional roles*. The only purpose of functional roles is to group permissions. In principal, users are not directly assigned to them.

A functional role is often refined into other roles, either other functional roles or user-roles, that are authorised to include its permissions. For example, in figure 3.4 the programmer's role is specialised into the roles of system, application, and real-time programmer.

Functional and user-roles can vary from location to location, and from project to project. Usually they contain a common set of permissions and have additional permissions that depend on their specific requirements. To define different roles for almost the same function or job position would be redundant and lead to administrative confusion. In order to avoid having many and unrelated role names for the same job functionality, *extensional roles* are created. Roles are extended on basis of location, branch or project where a particular function is fulfilled. For example, a functional role of a bank teller may have extensional roles depending on the branch: a bank teller_London and a bank teller_Cambridge; or the role of a system programmer may be involved in several projects and additional permissions would be assigned to the extensional roles.

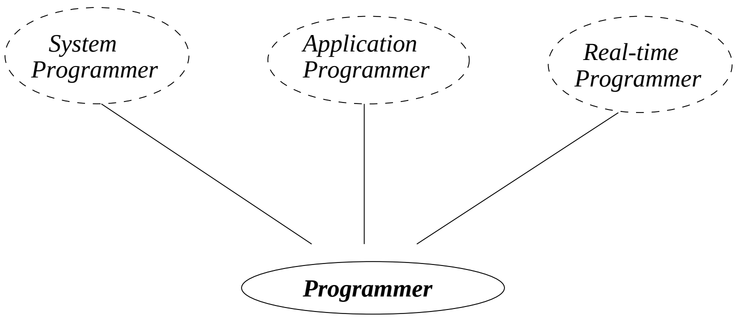

Figure 3.4: Specialisation of Roles.

3.4.2 Constructing the Role Hierarchy

In an organisation with a large number of diverse duty requirements, the number of roles increases as new roles are defined to meet specific needs. Some of these roles (usually a large number) will have overlapping functions, and hence overlapping permissions. Tracking the distribution of such permissions among roles can become a very complex and tedious task. In order to simplify administration and analysis of permission distribution, it is important to have some means for organising the various roles. Such structures of roles, defined by Baldwin [Bal 90] as privilege graphs and by Ferraiolo *et al* [FKC 95, MD 94, SCFY 96] as role hierarchies, are often discussed alongside roles.

In this section we describe the essential properties that must hold in a role organisation structure so as to offer a consistent permission distribution framework, and to provide the basis for enforcement of security constraints, that will be presented in the next chapters.

The main concept that role organisation utilises is *inheritance*, when one role inherits or includes permissions of other roles.

Definition 4 (Inheritance) *An inheritance relation $\longrightarrow$ is defined between two roles $\boldsymbol{r_i}$ and $\boldsymbol{r_j}$, denoted $\boldsymbol{r_i} \longrightarrow \boldsymbol{r_j}$, if and only if $\boldsymbol{PR(r_i)} \subseteq \boldsymbol{PR(r_j)}$.*

In this relation role r_j is seen as a *senior* role, whereas r_i is a *junior* role. In addition to directly assigned permissions role r_j indirectly contains the permissions of the junior role r_i. Thus, the function $PR(r)$ returns not only all direct permissions of the role, but also all indirectly contained (inherited) permissions. This combined set of permissions will be called *effective* permissions of the role.

To obtain directly assigned permissions of a role we in addition define function $dPR(r)$, such as: $\forall r \in R, \forall r_i : r_i \rightarrow r \;\; dPR(r) = PR(r) - PR(r_i)$.

The inheritance relation is:

reflexive $r_i \rightarrow r_i$ since $PR(r_i) \subseteq PR(r_i)$;

antisymmetric $((r_i \rightarrow r_j) \wedge (r_i \rightarrow r_j)) \Rightarrow r_i = r_j$
since $(r_i \rightarrow r_j) \Rightarrow PR(r_i) \subseteq PR(r_j)$
and $(r_j \rightarrow r_i) \Rightarrow PR(r_j) \subseteq PR(r_i)$;

transitive $((r_i \rightarrow r_j) \wedge (r_j \rightarrow r_k)) \Rightarrow (r_i \rightarrow r_k)$
since $(PR(r_i) \subseteq PR(r_j)) \wedge (PR(r_j) \subseteq PR(r_k)) \Rightarrow (PR(r_i) \subseteq PR(r_k))$.

From the latter property it follows that the function $PR(r)$ increases **monotonically** in respect to the inheritance ($\rightarrow$) relation among roles.

Roles are organised into role hierarchies that reflect the authority of functionality attached to each role. An example of such role hierarchy is presented in figure 3.5. The roles that inherit permissions from other roles have higher functionality and are more powerful. They are also higher in the hierarchy, represented on the top of less powerful junior roles. In the example from figure 3.5 both an application programmer and project_1 manager are senior to the role of a programmer. By inheriting the permissions of the junior roles that add up to its own access rights the role of a project_1 manager becomes associated with bigger authority.

The hierarchies of roles with inheritance relations among them are combined to form a *directed acyclic graph*[5], such that with the roles as nodes for a given directed edge $< r_i, r_j >$: $PR(r_i) \subseteq PR(r_j)$.

Definition 5 (Role Graph) *A role graph is an inheritance relation on the set of roles, defined as* $\boldsymbol{RG} = (\boldsymbol{R} \times \boldsymbol{R}, \longrightarrow)$. *The nodes are the roles from the set* $\boldsymbol{R}$*, and the edges are defined with* $\longrightarrow$ *relation.*

Although roles are organised to reflect an organisation's lines of authority and responsibility, this does not imply that they must be structured according to the established organisation's responsibility lines. The inheritance of permissions offer more powerful combination of access rights to the senior roles. The effective permissions of a role are those directly associated with it, and those indirectly available through junior roles. Because we required that the *principle of least privilege* be applied when creating roles, the same should be ensured when creating the inheritance relations among roles; no role should inherit more permissions than are necessary for its functional requirements. For example, although a manager of some department might be senior in authority than other roles of the same department it does not necessarily mean that this role should inherit the permissions from all roles junior in authority.

3.5 Authorisation State

The described elements and system functions (summarised in figure 3.6) form a static authorisation database of our security model. An authorisation database is created for each application of the security model indicating users, roles, user authorisation lists, and a role graph. *An authorisation state* is, therefore, a 4-tuple

$$(Users, Roles, UserAuthorisationLists, RoleGraph).$$

The elements here are not subject to dynamic changes. The transitions in the authorisation state can be triggered only by administrators performing administrative operations[6].

[5]Further reading on graph theory employed for role graph management can be found by Nyanchama and Osborn in [NyOs 94].

[6]More attention to the authorisation database and administrative operations is given in chapter 6.

The roles authorised to a user have to be activated, so that the user can use the associated permissions. We assume that the user's discretion alone determines which authorised roles are activated. An active role set is associated with each user, defined as the following mapping function:

$ARS(u) : U \rightarrow 2^R$, the user/active role mapping, which gives the set of roles active for user u.

This adds a dynamic element to the above authorisation state. The changes in it are triggered by users, dynamically activating and deactivating their authorised roles.

3.6 Access Granting Rules

We can now proceed to define the rules under which an access is granted to a user within our security model.

Firstly, a user has to activate one or more roles authorised to him.

Rule 1 (Role Activation) *A user can activate a role if and only if the user is authorised to this role.*

$$\forall u \in U, r \in R : u \in UR(r) \Rightarrow can\_activate(u, r). \tag{3.1}$$

Only after the roles are activated, the user can proceed to request access to objects and to perform permitted transactions. The user is able to use all permissions associated with activated roles, including inherited ones[7].

Rule 2 (Exercising a Permission) *A user can exercise a permission if and only if the permission is an effective permission of the user's active role.*

$$\begin{aligned}&\forall u \in U, p \in P :\\ &\quad \exists r \in ARS(u) : p \in PR(r) \Rightarrow can\_exercise(u, p).\end{aligned} \tag{3.2}$$

Since a permission was defined as the right to execute a transaction on specific object(s), the latter rule can be re-written to include authorisation to transaction and object(s) access.

Rule 3 (Object Access) *A user can perform a transaction on an object if and only if there exists a role in the user's active role set and this role has a permission authorising execution of the transaction that operates on the object.*

$$\begin{aligned}&\forall u \in U, tr \in Tr, obj \in Obj :\\ &\quad \exists r \in ARS(u) \land \exists p \in PR(r) : tr \in TrP(p) \land obj \in ObP(p)\\ &\quad \Rightarrow can\_execute(u, tr, obj).\end{aligned} \tag{3.3}$$

[7]In chapter 6 we consider how to restrict usage of inherited permissions in some cases.

3.7 Conclusions

This chapter described our basic access control framework that uses roles as means to combine several related permissions under the same name, by associating them with the business and job position. A number of different specifications of role-based access control has been previously given by various authors. Although the framework described here is not identical to any of them, it captures the essential features such as referenced by Sandhu *et al* [SCFY 96] and Ferraiolo *et al* [FKC 95].

In specifying a role we wanted to refer to the functional aspect that captures the responsibilities and rights pertaining to some duty requirements. A role was defined in terms of permissions and was viewed as a unit of access rights that exists separate from users. The inheritance relation between roles was presented also in terms of permissions, and not in terms of role membership by users as has been described in some models [FKC 95, FBK 98].

With respect to organisation-specific requirements, role-based access control allows for incorporation of application-level security constraints and semantics. In the next chapter, we aim to exploit the variety of constraints that can be enforced within this framework.

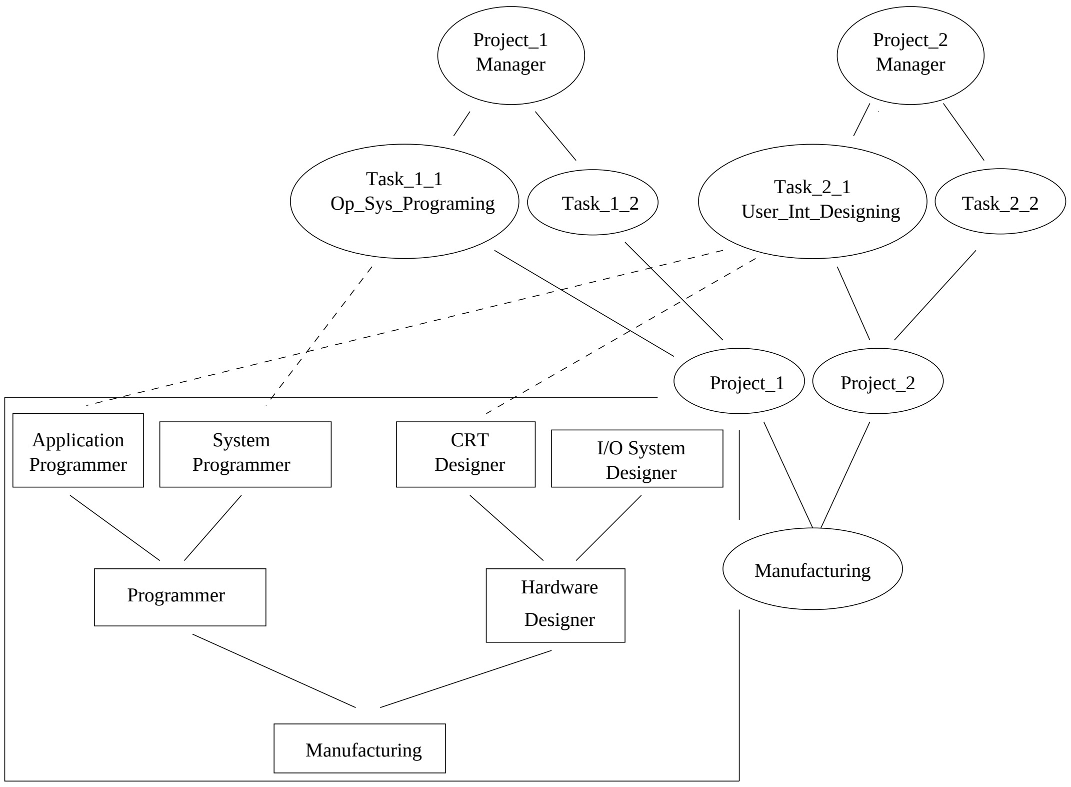

Figure 3.5: Example of Role Hierarchies.

- U = a set of users, $u_1, ..., u_n$;

  Tr = a set of transactions, $tr_1, ..., tr_n$;

  Obj = a set of objects, $obj_1, ..., obj_n$; and

  $P = Tr \times Obj$ a set of permissions, $p_1, ..., p_n$; and

  R = a set of roles, $r_1, ..., r_n$.

- $RG = (R \times R, \rightarrow)$ is a directed acyclic graph on R with edges defined by inheritance ($\rightarrow$) relation, such that

  $\forall (r_i, r_j)(r_i \rightarrow r_j) \in RG \Rightarrow PR(r_i) \subseteq PR(r_j)$.

- **user:** $UR(r) : R \rightarrow 2^U$, a function mapping role r to a set of users.

- **permissions:** $dPR(r) : R \rightarrow 2^P$, a function mapping role r to a set of permissions;

  permissions*: $PR(r) : R \rightarrow 2^P$, extends **permissions** to a set of all **effective permissions** in the presence of inheritance from other roles

  $\forall r \in R, \forall r_i : r_i \rightarrow r \;\; dPR(r) = PR(r) - PR(r_i)$.

- **transaction:** $TrP(p) : P \rightarrow Tr$, a function mapping permission p to one transaction.

- **object:** $ObP(p) : P \rightarrow 2^{Obj}$, a function mapping permission p to a set of objects.

Figure 3.6: Main Elements and System Functions.

Chapter 4

Security Constraints

4.1 Introduction

Previously, in chapter 2, we have recognised the existence of a wide range of requirements arising from organisational security policies. Separation of duty and the Chinese Wall policy were defined as two methods at the heart of commercial fraud and error control. Consistent enforcement of these security principles within an access control framework is as important as support for the least privilege principle, and as significant to commercial integrity as implementation of Bell-LaPadula's principles is to military non-disclosure policies.

In this chapter we present a framework for applying security policies of commercial organisations in the RCBS model. We specify constraints necessary for laying out the separation of duty policy and for simulating the Chinese Wall properties. We examine their implications for permission distribution in role hierarchy.

4.2 Separation of Duty Requirements

Clark and Wilson [CW 87] were first to call major attention for enforcing separation of duty within security systems. They described separation of duty as attempting to ensure the correspondence between data objects within a system and the real world objects they represent. Since this correspondence cannot normally be verified directly, it is ensured indirectly by separating all operations into several subparts and requiring that each subpart be executed by a different person. Clark and Wilson then proposed, that to implement separation of duty in computer systems each user must be permitted to use only certain sets of programs (transactions). This approach is later referred as static separation of duty policy that is realised at design time.

Since then several other definitions of separation of duty have been proposed, as enumerated in figure 4.1. Dynamic separation of duty introduced by Sandhu [San 90] provides greater flexibility by allowing a user to perform conflicting transactions, but only on distinct objects. Nash and Poland [NP 90] developed this concept further by introducing the notion of dynamic object-based separation of duty, which forced every transaction against an object to be executed by a different user.

"No user of the system, even if authorised, may be permitted to modify data items in such a way that assets or accounting records of the company are lost or corrupted. Essentially there are two mechanisms at the heart of fraud and error control: the well-formed transactions, and separation of duty among employees. The most basic separation of duty rule is that any person permitted to create or certify a well-formed transaction may not be permitted to execute it. This rule ensures that at least two people are required to cause a change in the set of well-formed transactions." (by Clark and Wilson [CW 87])

"Separation of duty is enforced by the rule that for transient objects different transactions must be executed by distinct users". (by Sandhu [San 90])

"It is a security principle used to formulate multi-person control policies, requiring that two or more different people be responsible for the completion of a task or set of related tasks. The purpose of this principle is to discourage fraud by spreading the responsibility and authority for an action over multiple people, thereby raising the risk involved in committing a fraudulent act by requiring the involvement of more than one individual." (by Simon and Zurko [SZ 97])

"Its purpose is to ensure that failures of omission or commission within an organisation are caused only by collusion among individuals and, therefore, are riskier and less likely, and that chances of collusion are minimised by assigning individuals of different skills or divergent interests to separate tasks." (by Gligor *et al* [GGF 98])

"In complex environments where the actions of ill-intentioned users can create financial or other damage to a company, it is common to identify combinations of operations which should not be authorised to a single user. Policies which deal with preventing fraud are called separation of duties policies." (by Nyanchama and Osborn [NyOs 99])

Figure 4.1: Definitions of Separation of Duty.

4.2.1 Taxonomy

Although separation of duty is intuitively easy to understand, so far there is no systematic and comprehensive approach for applying many variations of this principle in security models. Separation of duty (SoD) can be analysed as conflict of interest constraint along different dimensions: as conflict between roles, between operations, between objects, or between users.

Simon and Zurko [SZ 97] informally enumerated all the variations of separation of duties that have been mentioned in previous sources, while Gligor *et al* [GGF 98] presented formal definition of all best known to date properties of separation of duty. According to the taxonomy described in those papers, it is possible to distinguish the following forms of separation of duty:

- simple static/dynamic SoD

  a user is permitted to use only certain sets of operations;

- operational SoD

  no user is allowed to perform all operations forming a business task, regardless of target object;

- object-based SoD

  no user can be allowed to act upon the same object more than once;

- history-based SoD

  no user can be allowed to perform all operations forming a business task on the same object.

A simple static/dynamic separation of duty is considered in most role-based access control models, and is implemented by declaring certain roles as being mutually exclusive (Ferraiolo *et al* [FBK 98, FKC 95]). A common example is that of mutually disjoint organisational roles, such as those of purchasing manager and accounts payable manager. Generally, the same individual is not permitted to access both roles because this creates a possibility for committing fraud.

An operational separation of duty policy is used to prevent one person from doing all parts of a business function/task that should require two or more. A frequently used example in this case is a business function involving preparing and approving purchase orders. Presented in figure 4.2, this task consists of a number of steps, each with a different operation. For this task to be fulfilled, each step must be successfully completed. For it to be fulfilled according to an operational separation of duty policy, each step must be performed by a different user. Since it is not always practical to apply operational SoD on every existing business task, the most sensitive tasks must first be identified, and then SoD is applied only on the critical operations within those tasks. The rule of operational SoD then expresses the following:

no user can be allowed to perform all operations that form a critical combination of a business task.

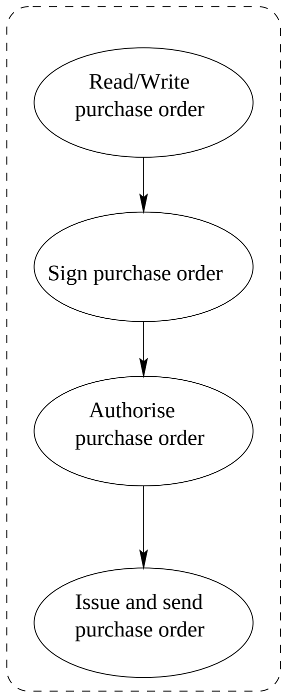

Figure 4.2: Example of a Business Task.

An object-based separation of duty policy is often subject to different interpretations, since the informal definition presented above [NP 90] does not specify precisely which objects and operations are subjected to this rule. One way to interpreting this rule is to restrict a user from performing an operation in a business task on an object if the user has already performed another operation of the business task on the same object, but this corresponds exactly to the history-based SoD viewed here as a distinct type of SoD. Another interpretation is to consider critical conflicts that might be existing among groups of objects. The purpose then is to prevent the same user from acting upon objects from conflicting groups.

We may recall from chapter 2, that the security principle of the Chinese Walls is applied to disallow people from accessing information that is held to conflict with any other information that they already possess. This principle and the latter variant of object-based SoD both have similar purpose, in the sense that information held in objects is distributed among different conflicting groups, and the rules are specified on how these groups are then accessed by the authorised users. We will review the properties of the Chinese Wall policy later in this chapter. We then propose how those properties can be simulated with constraints of object-based SoD.

A history-based separation of duty policy combines properties of both operational and object-based separation of duty. Simon and Zurko [SZ 97] described history-based SoD as expressing the most desirable properties of SoD policies. Object-based SoD does not allow a user to perform a second action on an object when this makes sense and is allowed by (human) policy, and operational SoD does not allow one user to perform all the actions of a task on different objects when this makes sense and is allowed by (human) policy. History-based SoD is used to restrict a user just from performing the operations of one business task on one object. The central part in this SoD policy is played by histories recording previous user accesses. To implement this policy, the mechanisms must be in place that record relevant history, and later analyse it.

In this chapter we will present a systematic framework for applying separation of duty policies in the RCBS model. Our work builds upon SoD properties analysed by Simon and Zurko [SZ 97], and formalised by Gligor *et al* [GGF 98]. In addition to the known properties, we identify other significant SoD properties which have not been previously defined. In particular, we emphasise the importance of permission-centric SoD constraints. We also include the notion of role hierarchies in SoD specifications, and introduce new SoD properties required for simulating Chinese Walls.

4.3 Mutual Exclusion of Roles

Within role-based access control models separation of duty has been implemented with mutual exclusion of roles [FBK 98, FKC 95, SCFY 96]. The term "mutual exclusion" usually has the meaning that some form of conflict exists between pair of roles. Users are allowed to access only one role from this pair.

When implemented using role exclusion rules, separation of duty can be analysed along two dimensions: *when mutual exclusion is applied*, and *the permissions to which it is applied*. Ferraiolo *et al* [FKC 95] defined two types of role exclusion in their formal model of RBAC, an *authorisation-time exclusion* and an *activation-time exclusion*, that depend on whether the mutual exclusion is applied at the time a user is authorised to a role, or at the role activation time.

In the next sections we aim to explore how the framework presented previously is implicated by mutual exclusion of roles. In order to do that, we first extend an authorisation state to include pairs of roles that are mutually exclusive:

$Excl : R \times R$ is the set of mutually exclusive role pairs $(r_i, r_j) \mid i \neq j$, that hold one of the following relations:
$strong\_excl(r_i, r_j)$,
role pairs of authorisation-time exclusion type,
$weak\_excl(r_i, r_j)$,
role pairs of activation-time exclusion type.

4.3.1 Authorisation-Time Exclusion

According to Ferraiolo *et al* [FKC 95] the authorisation-time exclusion requires that the following constraint is satisfied in user-role authorisations:

Constraint 1 (Authorisation-time Exclusion) *A user is authorised to a role if the role is not mutually exclusive with another role to which the user is already authorised.*

$$\begin{aligned} &\forall r_a, r_b \in R \\ &\quad strong\_excl(r_a, r_b) \Rightarrow UR(r_a) \cap UR(r_b) = \emptyset. \end{aligned} \tag{4.1}$$

4.3.2 Activation-Time Exclusion

A similar but weaker form of exclusion is to allow users to be authorised for roles that are mutually exclusive, but allow them to activate only one such role at a time. This form of exclusion is applicable only at the time a user activates a role. In respect to our framework, we specify it as an extension to the previously defined role activation rule:

Rule 1a (Activation-time Exclusion) *A user can activate an authorised role if the role is not mutually exclusive with any other role in the user's active role set.*

$$\forall u \in U, \forall r_a, r_b \in R$$
$$weak\_excl(r_a, r_b) \wedge r_a \in ARS(u) \Rightarrow r_b \notin ARS(u). \tag{4.2}$$

4.3.3 The Effects on Permission Distribution

Role exclusion constraints express the requirements of simple static/dynamic SoD, as defined by Clark and Wilson [CW 87]. This means that they do not directly ensure that requirements of operational or object based SoD are satisfied as well. Therefore, the complications arise when exploring distribution of permissions of mutually exclusive roles.

For example, suppose there are two roles $R1$ and $R2$, which are mutually exclusive. Role $R1$ has permissions a and b, while role $R2$ has permission c. In addition, there are other two roles $P1$ and $P2$, that respectively have permissions a and b each. It is possible for a user which is authorised to role $R2$ to be authorised also to both role $P1$ and role $P2$, and thus gain the same permissions as of roles $R1$ and $R2$ taken together.

The described situation can occur, because role exclusion did not place any restrictions on the distribution of permissions among roles. In principal, a statement by the role designer that two roles are mutually exclusive or conflict with each other is a very powerful statement. Since each role is defined as a collection of permissions, mutual exclusion may define either the case that all permissions of two roles conflict, or the situation when only a small part of the two roles conflict.

For example, say there are two roles $R1$ and $R2$ with the sets of permissions $\{a, b, c, d\}$ and $\{e, f, g, b\}$ respectively. Suppose the designer wants to declare these two roles as mutually exclusive, but really it is only permissions c, d that conflict with permissions e, f. The reason for this might be operational or object based SoD. Thus, role exclusion constraint in this case is too strong. The only solution in this case would be to create two additional roles, such as role $S1$ junior to role $R1$ with permissions c, d, and role $S2$ junior to role $R2$ with permissions e, f. Mutual exclusion can then be declared between these two roles.

More flexible approach to solve the above example is to allow for finer granularity by allowing application of permission-centric constraints that specify conflicts between two separate permissions.

4.3.4 Complete Exclusion

In the remainder of this chapter we will examine in more detail the requirements of operational and object-based SoD to realise permission-centric constraints. First, however, we have to define more precisely what we mean by mutual exclusion in terms of permission sharing.

We say that when two roles are defined as mutually exclusive, a user authorised to one of the roles cannot have access to any of the permissions which are also assigned to the other role. This implies, not only that a user cannot have access to permissions of the two mutually exclusive roles, but also that a user authorised to one role from a mutually exclusive pair cannot assigned to any other role that has at least one permission in common with the other role from the pair. This property is independent of authorisation-time or activation-time exclusion. We formalise it into the following constraint, which is a fundamental assumption underlying what follows[1]:

Constraint 2 (Complete Exclusion) *If two roles are defined as mutually exclusive then the user authorised to one of the roles must not have access to any permissions that are part of directly assigned permissions of the other role.*

$$\begin{aligned}
&\forall (r_a, r_b) \in Excl : r_a \neq r_b \\
&\quad {}^{a}\forall r_c \in R \ \ dPR(r_a) \cap PR(r_c) \neq \emptyset \Rightarrow dPR(r_b) \cap PR(r_c) = \emptyset \wedge \\
&\quad {}^{b}\forall r_d, r_e \in R : UR(r_d) \cap UR(r_e) \neq \emptyset \\
&\qquad dPR(r_a) \cap PR(r_d) \neq \emptyset \Rightarrow dPR(r_b) \cap PR(r_e) = \emptyset \wedge \\
&\qquad dPR(r_a) \cap PR(r_e) \neq \emptyset \Rightarrow dPR(r_b) \cap PR(r_d) = \emptyset.
\end{aligned} \tag{4.3}$$

The formal specification says that (a) no one role is allowed to have permissions of both roles from a mutually exclusive pair, and (b) no two roles with a common authorised user are allowed to have permissions of both roles from a mutually exclusive pair.

The constraints reflect a definition of a role given in this work; if a user cannot be authorised to a role, it means that the user cannot be given any permission from the set of permissions associated with this role. Given such assumption, constraints on mutual exclusion of roles should be used only when conflict exists between every permission of two given roles. If the mutual exclusion constraint is too strong, then either roles have to be redefined with a finer granularity or permission-centric constraints described in section 4.4 are better applicable.

4.3.5 Constraints on Hierarchies

The defined constraints have direct implications on relationships between roles in the role hierarchy. Let us consider the example from figure 4.3. Authorising a user to role $R1$ makes permissions $\{a, b\}$ that are directly assigned to this role available to the user. Since an inheritance relation exists between roles $R1$ and $S1$, this also makes permissions $\{c, d\}$ available to the user authorised to role $R1$. Suppose that roles $S1$ and $S2$ are defined as mutually exclusive. If the inheritance relation is established also between roles $S2$ and $R1$

[1]Our notion of complete exclusion is fundamentally different from the one used previously by Kuhn [Kuhn 97]. Their complete exclusion says that if any role is mutually exclusive with another role, then no permission of the first role is assigned to any other role. In our cases, the permissions of mutually exclusive can be shared between other roles, they just cannot be available to the same user.

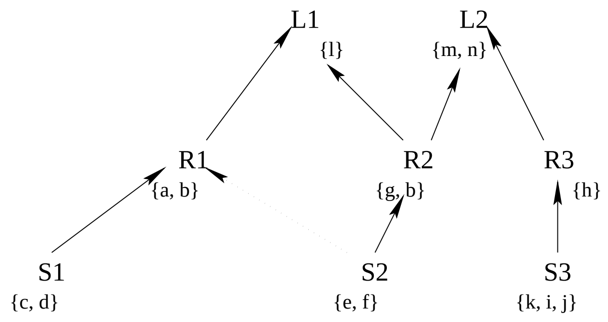

Figure 4.3: A Sample Role Hierarchy.

(a dotted line in figure 4.3), authorising a user to role $R1$ gives him permissions of both role $S1$ and role $S2$. As these roles have been declared mutually exclusive, this violates our complete exclusion constraint. Thus, declaring two roles as mutually exclusive has consequences to their common seniors. The following formalises this:

Theorem 1 *Mutually exclusive roles cannot have a common senior role.*

$$\forall (r_a, r_b) \in Excl \\ \nexists r_c \in R : r_a \rightarrow r_c \wedge r_b \rightarrow r_c. \tag{4.4}$$

$\triangleright$ Suppose there is a role r_c such that $r_a \rightarrow r_c \wedge r_b \rightarrow r_c$ and $(r_a, r_b) \in Excl$. The role r_c then contains permissions from both role r_a and r_b, and according to the constraint 2 is unauthorisable. Since there cannot exist unauthorisable roles, it follows that mutually exclusive roles must have no common seniors. $\triangleleft$

The next result is direct consequence of complete exclusion:

Theorem 2 *Mutually exclusive roles cannot inherit each other.*

$$\forall (r_a, r_b) \in Excl \quad r_a \not\rightarrow r_b \wedge r_b \not\rightarrow r_a. \tag{4.5}$$

Note that the defined mutual exclusion rules do not prevent two mutually exclusive roles from having a common junior role. This is important since many systems usually have one common role that all other roles inherit. This could be an "employee" role, the basic permissions of which are available by inheritance to other roles.

Mutually exclusive roles can still be inherited by other roles as long as they never have a common upper bound. This means that, when authorising a user to a role, we have to verify not only that this role does not conflict with already authorised roles, but also that there will be no conflicts among inherited roles.

Constraint 1a (Inheritance at Authorisation-time Exclusion) *A user is authorised to a role if the role does not inherit another role that is mutually exclusive either with the role to which the user is already authorised, or with the role junior to it.*

$$\forall r_a, r_b \in R, \forall r_c \in R : r_a \rightarrow r_c, \forall r_d \in R : r_b \rightarrow r_d, r_d \neq r_c \\ strong\_excl(r_a, r_b) \Rightarrow UR(r_c) \cap UR(r_d) = \emptyset \wedge \\ UR(r_b) \cap UR(r_c) = \emptyset \wedge UR(r_a) \cap UR(r_d) = \emptyset. \tag{4.6}$$

Similarly, the same applies to activation-time exclusion:

Rule 1b (Inheritance at Activation-time Exclusion) *A user can activate an authorised role if the role does not inherit another role that is mutually exclusive either with the already activated role, or with the role junior to any of the already activated roles.*

$$\begin{aligned}&\forall u \in U, \forall r_a, r_b \in R, \forall r_c \in R : r_a \rightarrow r_c, \forall r_d \in R : r_b \rightarrow r_d, r_d \neq r_c\\&\quad weak\_excl(r_a, r_b) \wedge r_a \in ARS(u) \vee r_c \in ARS(u) \Rightarrow\\&\quad r_b \notin ARS(u) \wedge r_d \notin ARS(u).\end{aligned} \tag{4.7}$$

4.4 Critical Combinations of Operations

When only some parts of two roles conflict, other types of constraints have to be introduced. They represent permission-permission conflicts, indicating that for whatever reason two or more permissions should never be authorised together to the same user.

In order to define constraints that indicate permission-permission conflicts, we first consider properties of operational SoD. We recall the example describing the business task of processing purchase orders. The structure or plan of this task can be seen from figure 4.2 as composed of four consecutive steps. According to the requirement of operational SoD each of these transactions must be executed by different user. This means, that the user who wrote the purchase order can never be permitted to authorise it.

To identify what transactions operational SoD is applied on, we introduce the concept of a *critical combination.*

Definition 6 (Critical Combination) *A critical combination defines the set of transactions of a task, where each transaction should be performed by a different user.*

Let T be the set of tasks[2]. Then a critical combination for the given task t can be obtained with the function $CC(t) : T \rightarrow 2^{Tr}$.

Within our role-based framework permissions of a role specify which transactions are available to an authorised user. If both transactions *write purchase order* and *authorise purchase order* are allocated to the same role, it means that they are also both available to the same user. Since this is not acceptable with our definition of operational SoD, one of the constraints must restrict transactions of the same business task from being allocated to one role.

Constraint 3 (Per-Role Operational SoD) *A role has permission to a transaction only if this transaction is not in a critical combination with another transaction already allocated to the role.*

$$\begin{aligned}&\forall r \in R, \forall p_i, p_j \in PR(r), \forall t \in T\\&\quad \{TrP(p_i), TrP(p_j)\} \not\subseteq CC(t) \vee TrP(p_i) = TrP(p_j).\end{aligned} \tag{4.8}$$

[2]The precise definition of tasks will be presented in the next chapter.

A user can still gain access to all transactions of the task, when the combined permissions of several roles have all critical transactions, and if the user is authorised to all these roles. For example, if the transaction *write purchase order* is accessible through the role $R1$, and the transaction *authorise* through the role $R3$, then the user can gain access to both transactions if he is authorised to both role $R1$ and role $R3$. Additional constraint must, therefore, ensure that some combinations of roles do not allow a user to have access to the transactions restricted by operational SoD.

Constraint 4 (Static Operational SoD) *A user is authorised to a role only if no transaction allocated to this role is in a critical combination with any transaction of any other role already authorised to the user.*

$$\begin{aligned}
&\forall r_a, r_b \in R : UR(r_a) \cap UR(r_b) \neq \emptyset \Rightarrow \\
&\quad \forall p_i \in PR(r_a), \forall p_j \in PR(r_b), \forall t \in T \\
&\quad \{TrP(p_i), TrP(p_j)\} \not\subseteq CC(t) \vee TrP(p_i) = TrP(p_j).
\end{aligned} \tag{4.9}$$

Since by inheritance all permissions of junior roles are authorised to a user when a senior role is authorised, there cannot be any relationships in role hierarchy between roles that have conflicting permissions. This results in the following security property:

Theorem 3 (Inheritance in Operational SoD) *Two roles r_a and r_b cannot be connected within a role graph if their combined set of permissions authorises transactions in a critical combination.*

$$\begin{aligned}
&\forall r_a, r_b, r_c \in R \\
&\quad \exists p_i \in PR(r_a), \exists p_j \in PR(r_b), \exists t \in T : \\
&\quad TrP(p_i) \in CC(t) \wedge TrP(p_j) \in CC(t) \wedge TrP(p_i) \neq TrP(p_j) \\
&\quad \Rightarrow (r_a \not\rightharpoonup r_b \wedge r_b \not\rightharpoonup r_a) \wedge (r_b \not\rightharpoonup r_c \vee r_a \not\rightharpoonup r_c).
\end{aligned} \tag{4.10}$$

The dynamic variant of operational SoD can be implemented by requiring that a user does not at the same time activate the roles that allow to perform transactions in a critical combination. In an actual implementation, however, and especially when users are allowed to open a number of sessions at the same time this requirement is not sufficient; even if a user may not be able to perform a transaction in one session, he can still perform it in a different session with different active role set. In chapter 6 we will consider how to control activation of mutually exclusive roles, when there are multiple sessions. In chapter 5 we also demonstrate that requirements of dynamic operational SoD could be better addressed with history-based SoD constraints.

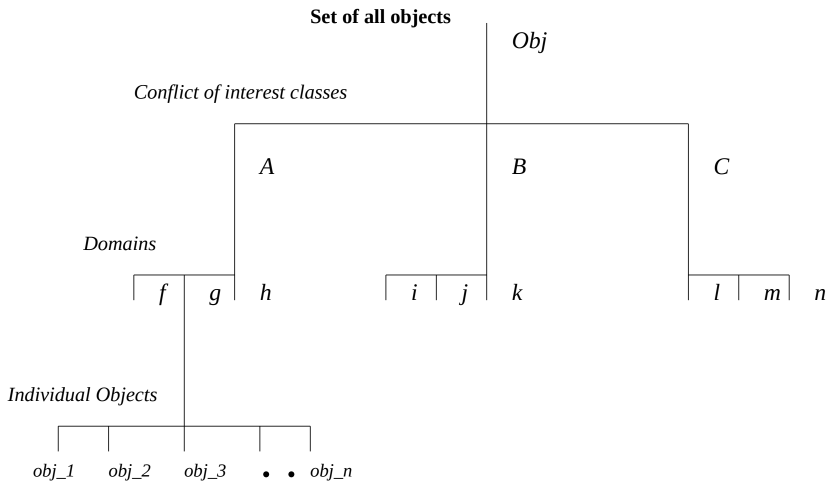

Figure 4.4: Composition of Objects.

4.5 Conflicting Datasets

In section 4.2 we have indicated that object-based SoD policy can apply to situations that require control of access according to conflict of interest between different sets of data. The sets of conflicting data may be the information about competitive corporations held within the same computer system in the organisation that provides corporate business services. A financial analyst within that organisation cannot at the same time advice to corporations that are in competition with each other. This means, that the sets of data on each corporation in competition are defined as being in conflict of interest class, and a single user is allowed to access at most one set of data within that class.

The above specification of object-based SoD is useful for simulating properties of the Chinese Wall policy in role-based access control. In the next section we describe this policy by adapting the description presented by Brewer and Nash [BN 89]. We then proceed to define object-based SoD constraints.

4.5.1 Chinese Walls

To describe the Chinese Wall policy it is necessary that all corporate information is stored in a hierarchical order with three levels of significance:

1. at the lowest level there are individual items of information, equivalent to what we have termed as *objects*;

2. at the next level objects are grouped together into *domains* under some selected attribute of the data (for example, only data that concern the same corporation);

3. at the highest level, domains are grouped into *conflict of interest classes*. The domains that are within the same class are viewed as conflicting.

The described composition of objects is illustrated in figure 4.4. Examples of conflict of interest classes could be the business sector headings, such as Financial Institutions, Oil Companies, while domains would be the names of individual companies listed under those headings.

Formally, we can specify that each object $obj \in Obj$ is associated with two security labels (x, y): function $X(obj)$ returns conflict of interest class x of the given object obj, while function $Y(obj)$ gives its domain y. The properties associated with those characteristics are as follows:

- $\forall obj_i, obj_k \in Obj : Y(obj_i) = Y(obj_k) \Rightarrow X(obj_i) = X(obj_k)$

  if two objects obj_i and obj_k belong to the same domain, then they also belong to the same conflict of interest class;

- $\forall obj_i, obj_k \in Obj : X(obj_i) \neq X(obj_k) \Rightarrow Y(obj_i) \neq Y(obj_k)$

  if two objects obj_i and obj_k belong to different conflict of interest classes then they must belong to different domains.

The basis of the Chinese Wall policy is that users are only allowed to access information which is not held to conflict with any other information that they already possess. This means, that if a user has been allowed to access an object from a domain belonging to some conflict of interest class, a Chinese Wall is created for that user around that domain. The wrong side of the Wall would be any domain within the same conflict of interest class as that domain within the Wall. A user can only be allowed to access objects within the same domain or belonging to different conflict of interest classes.

This policy is defined with the following security rule:

access is only granted if the object requested: (a) is in the same domain as an object already accessed by that user, or (b) belongs to an entirely different conflict of interest class.

4.5.2 Object-based SoD Constraints

In our framework the defined security rule can be enforced by specifying permissions not only in terms of transactions, but also in terms of object types, with constraints on instances.

We may recall the function $ObP(p) : P \rightarrow 2^{Obj}$, which gives the set of objects associated with the given permission p.

The permissions of a role regarding access to objects are determined at design time, and are not allowed to be changed at run-time. This means, that initially, i.e. before any user has requested activation of any role and access to any object, roles have to be correctly initialised. The permissions of the same role can only authorise transactions on the objects that are

- in the same domain,
- or within a different conflict of interest class.

We can ensure the above with the following security constraint:

Constraint 5 (Per-Role Object-based SoD)

$$\begin{aligned}&\forall r \in R, \forall p_i, p_j \in PR(r),\\ &\quad \forall obj_i \in ObP(p_i), \forall obj_j \in ObP(p_j)\\ &\quad Y(obj_i) = Y(obj_j) \vee X(obj_i) \neq X(obj_j).\end{aligned} \tag{4.11}$$

The next step is to control how users are assigned to roles that have permissions on conflicting datasets. Since a single user is not limited to one role, the user can still gain access to domains that are in conflict through some combination of roles.

Constraint 6 (Static Object-based SoD) *A user is authorised to a role only if the objects that can be accessed with permissions of this role are in the same domain or within different conflict of interest class than objects accessible with permissions of all other roles already authorised to the user.*

$$\begin{aligned}&\forall r_a, r_b \in R : UR(r_a) \cap UR(r_b) \neq \emptyset \Rightarrow\\ &\quad \forall p_i \in PR(r_a), \forall p_j \in PR(r_b), \forall obj_i \in ObP(p_i), obj_j \in ObP(p_j)\\ &\quad Y(obj_i) = Y(obj_j) \vee X(obj_i) \neq X(obj_j).\end{aligned} \tag{4.12}$$

As previously, the defined constraints have direct implications on inheritance relations between roles in the role hierarchy:

Theorem 4 (Inheritance in Object-based SoD) *Two roles r_a and r_b cannot be connected within a role graph if their combined sets of permissions gives access to objects from the same conflict of interest class but different domain.*

$$\begin{aligned}&\forall r_a, r_b, r_c \in R\\ &\quad \exists p_i \in PR(r_a), \exists p_j \in PR(r_b), \exists obj_i \in ObP(p_i), \exists obj_j \in ObP(p_j) :\\ &\quad X(obj_i) \neq X(obj_j) \wedge Y(obj_i) = Y(obj_j)\\ &\quad \Rightarrow (r_a \not\rightarrow r_b \wedge r_b \not\rightarrow r_a) \wedge (r_b \not\rightarrow r_c \vee r_a \not\rightarrow r_c).\end{aligned} \tag{4.13}$$

The dynamic variation of the described policy can be enforced only by monitoring the history of the previous accesses by users. The mechanisms to monitor and analyse those histories will be presented in chapter 5.

4.6 Other Security Constraints

In addition to constraints enforcing SoD policies, other security constraints may be required by a particular application. For example, a role may have prerequisite conditions that limit the ability of users to be authorised for the role or to activate the role. It may be the case that some roles in an application are accessible only if not simultaneously in use by other users, or that certain roles can be activated only from some particular terminal. Such entry conditions can be introduced either as static constraints or as activation rules, to be enforced either at authorisation or at activation time. In the next sections we briefly describe two of the most commonly used role entry conditions.

4.6.1 Conflicts Among Users

The conflicts of interest arising among different users are called as *user incompatibility.* This might be either the case when two or more users cannot be authorised to the same role, or the case when they cannot be active in the role at the same time. For example, the role of purchasing clerk is authorised to four users: John, Sarah, Tim, and Jane, but only John and Sarah or Tim and Jane can activate this role.

The conflict in this example can be resolved by first identifying which users authorised to the same role are incompatible. We define the function $IncomU(r, u) : R \times U \rightarrow 2^U$, that returns users authorised to the role r incompatible with the user u. Then the following condition has to be satisfied at every attempt to activate this role:

Rule 1b (Activation-time User Conflict) *An authorised user can activate the role if the user is not incompatible with any other user already active in this role.*

$$\begin{aligned} &\forall r \in R, \forall u_i, u_j \in UR(r) \\ &\quad u_i \in IncomU(r, u_j) \wedge r \in ARS(u_j) \Rightarrow \neg can\_activate(u_i, r). \end{aligned} \tag{4.14}$$

We must note, that in large distributed computer systems it is cumbersome to enforce user incompatibility at run-time, since it requires finding if the role in question has been activated by other users anywhere in the system. The situation could be helped by also placing restrictions on the place, indicating where the role should be activated.

4.6.2 Cardinality

Another constraint associated with the same difficulties at its run-time enforcement is the *cardinality* of a role. It requires that no more than a defined number of users are either authorised or active in the role. For example, only one person is allowed to be active in the role of a managing director at any given time, even though the role might be authorised to several users.

In this case a role has to be associated with a constant indicating the maximum number of users that can activate the role at the same time. Whenever a request to activate the role is presented, a system must verify that the associated constant is not exceeded. Again, this is more efficient if only a limited number of active sessions are opened.

Before a role can be activated an authorised user may be required to satisfy many different prerequisite conditions. For example, in order to access the role of a Chair of a meeting, a user is required first to have successfully activated the role of a Fellow, and also be a senior fellow to all other users that activated the role of a Fellow. This requires verifying current activations of the additional role, and also information about personal details of involved users.

4.7 Conclusions

In this chapter we have explored how constraints arising from the requirements of commercial separation of duty policies can be implemented within our role-based access control framework. We showed that it is possible to utilise role-based protection for enforcing the essential principles of many commercial security policies.

Specification of such constraints reduces complexity in ensuring consistency and completeness of organisational security requirements in applications. The constraints can be tailored to fit a specific application.

Chapter 5

Context-Based Authorisations

5.1 Introduction

The framework and security constraints presented in the previous chapters have been specified with respect to users, roles and permissions, and granting access rights to users. We studied the propagation of permission authorisation, but did not tie it with any activity in the system. The predefined authorisations to roles and permissions often allow access for more time than required; even if a user completes a task, the user still has the same permissions, until an administrator has not explicitly removed them. This results in compromised security.

In order to ensure that users possess authorisations only when required, a dynamic authorisation management framework has to be introduced. In this chapter we describe an approach to *active* authorisation control. The abstractions and mechanisms will be introduced that provide for the run-time management of access control, which is in accordance with emerging context associated with progressing tasks and operation sequences.

5.2 Background

First we discuss influence behind "active" access control, and present the main concepts used in this approach.

5.2.1 The Subject-Object View

In most security models a static approach to enforcing access control is implemented. Subjects, objects, and rights that subjects possess to gain access to the various objects are represented in such structures as capabilities or access control lists, that are defined by security administrators or designers, and that do not incorporate any dynamic changes. This static subject-object view of access control can be traced to the earliest security models. The access control framework described in chapter 3 has also been influenced by the same static view of access control; the rights users can gain are determined by analysing user authorisation lists associated with roles and permissions assigned to those roles. However, this view of access control has the following limitations:

- access control is detached from larger operational context;

- there is no association with past accesses;
- there is no record of usage of permissions;
- existing permissions can be revoked but cannot be suspended.

Access control in “passive” security models is enforced according to a simple rule which can be stated that *if a subject has requested an access operation to an object, and the subject possesses the permission to the operation, then access is granted.* No other contextual information about ongoing tasks is taken into account when evaluating access requests; so long as a permission exists the subject can use it any number of times.

5.2.2 “Active” Access Control

The preliminary ideas for an “active” access control were presented by Thomas and Sandhu [TS 93, TS 94]. An objective of this novel approach is to ensure that access control decisions correspond to the current business activities.

A cursory look at an organisation reveals a complex web of activities that often span different departments. One can view these activities as composed of a number of tasks initiated by users in accordance with their responsibilities and duties in the organisation. When task execution is computerised, as in a workflow technology, we are faced with the problem of maintaining the required integrity. One concern is to ensure that operations in tasks are performed only by authorised users. Another concern is how to make sure that authorised users use these operations to perform only the tasks, required by organisation’s current business activities.

The latter can be achieved if access requests are being evaluated within the context of progressing tasks; access to the specific operation is granted only if this operation is required to complete a started task. This results in permissions being enabled and disabled according to authorisation requirements within context of tasks. Context-based authorisations are concerned with evaluating previous accesses and planning future access decisions. To illustrate the role of context-based authorisations let us consider the co-ordinated execution of tasks in the purchase process, shown in figure 5.1.

5.2.3 Processing of Tasks

The goods purchase process consists of a number of tasks, including preparation of requisition orders, receiving of goods, and issuing payments. During the preparation of the requisition order, clerks are given permissions to *create* (read/write) and *sign* the order, and the order becomes a valid document only after its is countersigned by an approving supervisor. Once the order is sent for approval the *write* permission on this order of the clerk role has to be disabled. In addition, for any given order, the operations *create* and *approve* must be performed by different users.

In “active” access control, not all permissions that have been allocated through access control lists, or user authorisation lists and roles, can actually be used at any moment in the system. A permission must in addition be authorised according to the context, based on previous accesses and current activities within tasks.

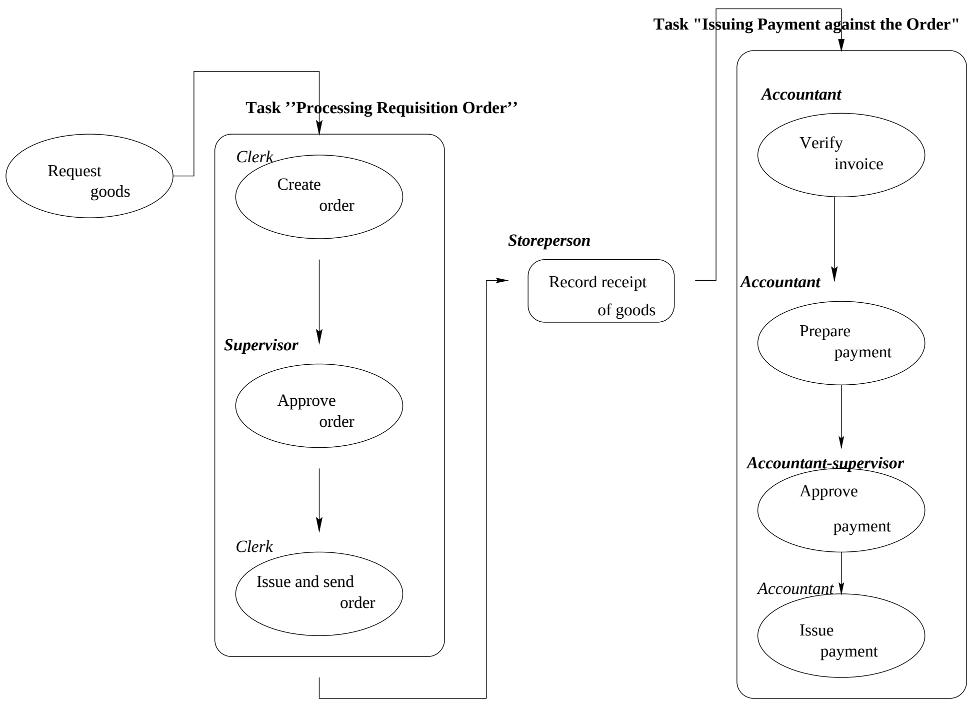

Figure 5.1: The Purchase Processing.

5.2.4 Related Work

The work towards specifying notions that support active access control and provide just-in-time permissions has been two-directional. Thomas and Sandhu [TS 97] developed a concept of an authorisation (step) granted while processing the tasks. Their authorisation (step) models the equivalent of a single act of granting a signature in the paper world. It is associated with rich semantics, such as going through a number of states during its lifetime. Each state of an authorisation (step) implies that certain permissions are granted while others are put on hold.

Atluri [AH 96] and Bertino [Ber 99, Ber 97] specify authorisations in association with workflow management systems. Since workflows represent processes that consist of several well-defined tasks, the authorisation flow can be synchronised with the workflow.

The general motivation behind both approaches is similar to ours in that they try to synchronise authorisations with the execution of tasks and so provide just-in-time permissions. However, we concentrate only on how "active" control can be applied within a role-based security framework. In presenting context-based authorisations we want to extend access control framework defined in chapter 3.

5.3 Task-based Authorisations

In this section we describe the components that make up a task. Based on definitions of tasks and task authorisation templates, we will then proceed to present a framework for enabling the permissions of roles within the context of execution progress in task instances.

5.3.1 The Construction of Tasks

A task t can be defined as a set $trset$ of a partial or total order of transactions $\{tr_1, tr_2, ..., tr_n\}$, that involve manipulation of objects. The execution of the task consists of processing relevant objects by subjects (transactions). Thus, every task starts when one or more objects have to be processed within the task. For example, the processing of an order is initiated with the request for some goods. In this task, the requisition order against the requested goods is created, approved, and sent to the appropriate vendor. It is possible to assume that each step in a task starts when one or more objects arrive and finishes when object leaves [1].

At the time a task is defined, however, only the types of objects to be processed are known. If Obj denotes the set of objects, then $Type$ is introduced to represent a finite set of object types. We can define a function:

$type(obj) : Obj \rightarrow Type$; if $type(obj_i) = type_i$, then obj_i is of type $type_i$.

Now we can give the following definition of a task:

Definition 7 (Task) *A task **t** is defined as a 3-tuple*

$$< \boldsymbol{task\_trset}, \boldsymbol{task\_typeset}, \boldsymbol{plan} >,$$

*where **task_typeset** $\subseteq$ **Type** is the set of object types that are allowed to be processed in the task, **task_trset** is the set of transactions to be performed in task **t**, and **plan** specifies the order in which transactions have to be executed in the task, given in the form of predicate-based rules.*

Accordingly, we introduce the following mapping functions:

$TrT(t) : T \rightarrow 2^{Tr}$, gives the transactions in the given task t,

$TypeT(t) : T \rightarrow 2^{Type}$, gives the set of object types that can be processed in the given task t.

Whenever a task is being executed, a task instance t_inst is generated. Any task may have several instances. We define a function:

$TInst(t) : T \rightarrow 2^{T\text{-}INST}$, that gives the set of task instances of the given task t.

In the task instance the concrete objects are being processed of the types given in $TypeT(t)$. The execution of each task instance is performed according to the plan specified in the task definition.

We assume that a special planner exists in a computer system that suggests the sequence the transactions have to be executed in each task instance. Implementation of such planner can be found in the workflow management systems (Georgakopoulos [GH 95]) and business process models (Holbein *et al* [HTB 96]).

[1] In the workflow systems each task step goes through the various processing states within a life-cycle. However, in presenting task definition within our framework we take a simplified approach.

5.3.2 Authorisation Templates

Since processing of a task involves performing transactions on objects, it means that certain permissions of certain roles will be used by authorised users. The permissions must be valid only during a lifetime of a task; once the task is finished the permissions must be no longer used, i.e. they have to be revoked or suspended. We have to know in advance what permissions of what roles are authorised for each step in a task.

In our framework, for any given task there exists an authorisation template[2]:

Definition 8 (Authorisation Template) *A task authorisation template* $\boldsymbol{AT}$ *is defined as a finite set of triples* $< \boldsymbol{p_i, r_i, (type_i, -)} > |\boldsymbol{i} \in \{\boldsymbol{1, ..., n}\}$*, where* $\boldsymbol{p_i}$ *denotes the permission of role* $\boldsymbol{r_i}$ *to be enabled for executing transaction in the task:* $\boldsymbol{TrP(p_i)} \in \boldsymbol{TrT(t)}$*, and* $\boldsymbol{(type_i, -)}$ *is an object hole that can be filled only by the object of type* $\boldsymbol{type_i}$*.*

An authorisation template gives reference as to what permissions of which roles can be used during execution of a task. When a task starts the permissions of roles defined in a template are enabled for the objects of the indicated type. Users that can use these permissions are not defined in an authorisation template, since the user's access to permissions is controlled via roles. Users must activate necessary roles in order to execute transactions of a task.

5.3.3 Authorisation Tokens

Whenever an execution of a transaction is scheduled by the planner, an authorisation token is generated to enable a permission of some role.

Definition 9 (Authorisation Token) *A permission authorisation token for each enabled permission* $\boldsymbol{p}$ *in a task instance* $\boldsymbol{t\_inst}$ *is defined as a 7-tuple* $< \boldsymbol{p, obj, r, t\_inst, usage\_count, denied\_users, executors} >$*.*

The authorisation token indicates that the permission $\boldsymbol{p}$ of the role $\boldsymbol{r}$ can be used on object $\boldsymbol{obj}$ in a task $\boldsymbol{t\_inst}$ a $\boldsymbol{usage\_count}$ defined number of times.

In the definition, *denied_users* is a list of user names that should not be allowed to use the enabled permission. *executors* represent users that have already used the permission. This component is filled in only when the permission is being used. We will explain later the purpose of having those attributes in an authorisation token.

An authorisation token indicates that a certain permission of some role has been enabled, and can be used in a task instance. Values in the token are derived from the task authorisation template, after the execution of the corresponding transaction has been scheduled by the planner.

The usage count is associated with each enabled permission. Its value is determined while defining the task authorisation template. In most cases this would be equal to 1. The count is reduced each time the permission is being used. When it reaches zero, the authorisation token becomes invalid. This means that the permission can no longer be

[2]The notion of templates is also used by Atluri [AH 96]. Our notion of authorisation templates is different in the sense that it is used only as a reference to enable the correct permissions.

used in this particular task instance. Such approach enforces constant and automated check-in and check-out of permissions in our framework.

If $Auth\_Tokens$ denotes the universal set of authorisation tokens, then we can define a function $AToken(t\_inst) : T\_INST \rightarrow 2^{Auth\_Tokens}$, that gives the set of authorisations tokens generated to enable permissions in a task instance t_inst.

For each authorisation token we also define the following mapping functions:

$PAuth\_Token(at) : Auth\_Tokens \rightarrow P$, gives the permission enabled with authorisation token at;

$RAuth\_Token(at) : Auth\_Tokens \rightarrow R$, gives role name for which the permission is enabled;

$OAuth\_Token(at) : Auth\_Tokens \rightarrow Obj$, gives the object on which the action is authorised;

$CountAuth\_Token(at) : Auth\_Tokens \rightarrow N$, gives current value of usage count in the given authorisation token at;

$Executors(at) : Auth\_Tokens \rightarrow 2^U$, gives users that finally used the enabled permission.

When an authorisation token is initially generated the set of executors is empty, but fills in whenever an enabled permission is being used. The user using the enabled permission must be authorised either to the role from the token, or to the role senior to it.

Property 1 (User of Enabled Permission) *A permission enabled in an authorisation token can be used only by a user that is authorised to the role defined in the token, or to the role senior to it.*

$$\begin{aligned} &\forall at \in Auth\_Tokens, \forall u \in Executors(at) \\ &\quad u \in UR(RAuth\_Token(at)) \vee \\ &\quad \exists r \in R : (RAuth\_Token(at) \rightarrow r) \wedge u \in UR(r). \end{aligned} \tag{5.1}$$

5.3.4 Enabled and Disabled Permissions of Roles

Conceptually, we can think of permissions associated with roles as being enabled and disabled according to authorisations in task instances. Previously, permissions were allocated to roles according to a broad policy defined to reflect access requirements and restrictions. Context-based access control, on the other hand, is used to manage permissions according to the requirements of current progressing activities, by associating the dimension of tasks with access control mechanisms.

The total set of permissions that is associated with a role can be seen in figure 5.2 as composed of permissions that have been enabled for specific tasks, and of other permissions that, though assigned to a role, cannot be used, because they are either disabled or have not yet been enabled within the context-based authorisation framework. Each role has a set of assigned permissions, and within that a set of enabled permissions.

We define a function $Enabled\_PR(r, obj) : R \times Obj \rightarrow 2^P$, that gives enabled permissions of role r that can be used on object obj.

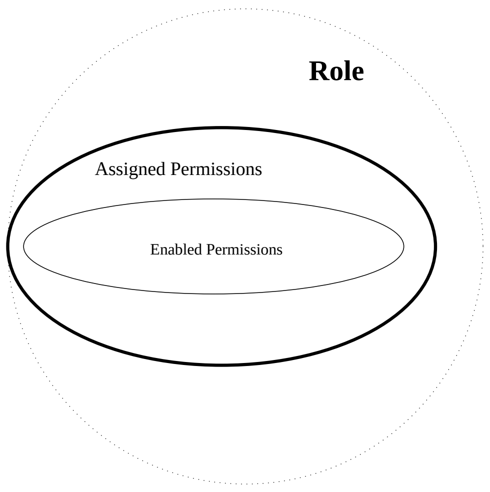

Figure 5.2: Permissions of a Role.

Property 2 (Enabled Permissions of Roles) *A permission is an enabled permission of role* ***r*** *on object* ***obj*** *if there exists a valid authorisation token within some task instance that authorises the permission of the role to be used on the object.*

$$\begin{aligned} &\forall r \in R, \forall p \in PR(r), \forall obj \in ObP(p) \\ &\quad p \in Enabled\_PR(r, obj) \Rightarrow \\ &\quad \exists t \in T, \exists t\_inst \in TInst(t), \exists at \in AToken(t\_inst) : \\ &\quad p \in PAuth\_Token(at) \wedge r \in RAuth\_Token(at) \wedge \\ &\quad obj \in OAuth\_Token(at) \wedge CountAuth\_Token(at) \neq 0. \end{aligned} \tag{5.2}$$

Figure 5.3 illustrates this property.

5.3.5 History-Based Separation of Duty

In presenting the above task-based authorisation framework, we said that permissions are enabled and appropriate authorisation tokens are generated for each task instance according to the information from a task authorisation template. It is also possible to specify certain constraints that apply conditions on what roles are selected for the enabled permissions, and what users are allowed to use them. A framework for enforcing some of such constraints is presented by Bertino *et al* [Ber 97]. However, the constraints of separation of duty policies, that are of particular importance in our security model, have not received much attention.

According to the taxonomy of separation of duty policies described in chapter 4, two variations of this policy are concerned with tasks; one of them being operational SoD, and another - history-based SoD. In the previous chapter we have specified how constraints pertaining to static operational SoD can be applied in a framework of role-based access control which is not tied with any context of current or previous activities. Both dynamic

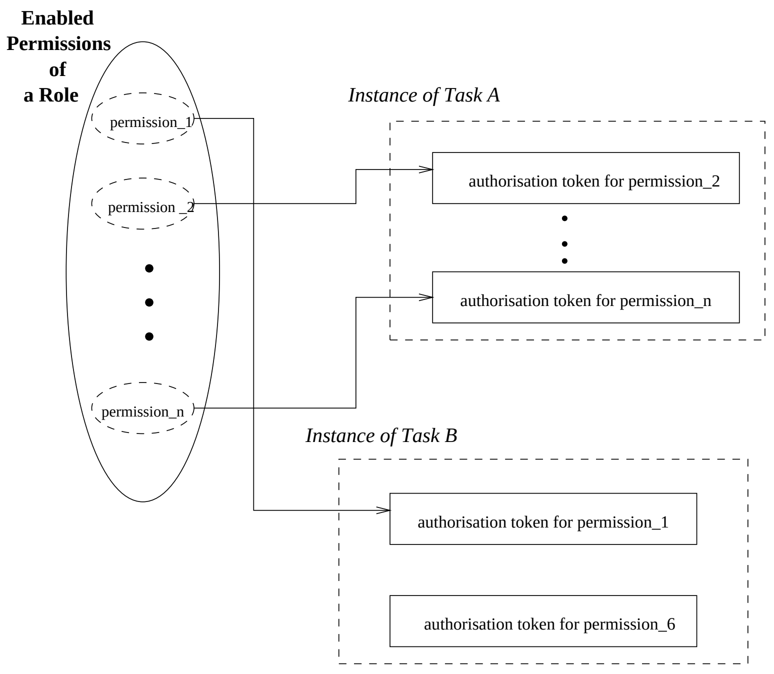
Enabled Permissions of a Role
permission_1
permission _2
permission_n
Instance of Task A
authorisation token for permission_2
authorisation token for permission_n
Instance of Task B
authorisation token for permission_1
authorisation token for permission_6

Figure 5.3: Enabled Permissions.

operational Sod and history-based SoD, on the other hand, require to take into account the current status of activities that have been performed within tasks.

In operational SoD policy, no user is allowed to perform all operations that are in a critical combination of some task. In history-based SoD policy, no user is allowed to perform all operations that are in a critical combination of a task on the same object in any instance of this task. The operational SoD can be viewed as based on a general constraint for a task, while the history-based SoD applies to every task instance.

In the next section we present how separation of duty properties can be realised within the framework of context-based authorisations. The constraints could be viewed as enforcing both history-based SoD, and dynamic operational SoD policies.

5.3.6 Denied Users

We require that a user who performed one operation from a critical combination should not be allowed to perform another operation from this combination within the same task instance. For example, the user that created the requisition order in the order processing task cannot be given permission to approve the same order. This means that for the task instance where this order is being processed, an authorisation token of permission "approve the order" must indicate that this permission cannot be used by the same user that created the order.

We may recall the attribute $denied\_users$, that was specified in the definition of an authorisation token. The users that performed other transactions from a critical combination have to be recorded in this set of denied users, if the transaction of enabled permission is in critical combination with these other already performed transactions.

A mapping function $Denied\_Users(at) : Auth\_Tokens \rightarrow 2^U$ gives the set of users that are denied use of permission $PAuth\_Token(at)$, enabled with authorisation token at.

Constraint 7 (Denied Users) *A user that performed a transaction from a critical combination cannot use the permission that allows to perform another transaction from the critical combination in the same task instance.*

$$\begin{aligned}
&\forall t \in T, \forall t\_inst \in TInst(t), \forall at_i, at_j \in AToken(t\_inst)\\
&\quad Executors(at_i) \neq \emptyset \wedge Executors(at_j) = \emptyset \wedge\\
&\quad TrP(PAuth\_Token(at_i)) \in CC(t) \wedge\\
&\quad TrP(PAuth\_Token(at_j)) \in CC(t) \wedge\\
&\quad TrP(PAuth\_Token(at_i)) \neq TrP(PAuth\_Token(at_j))\\
&\quad \Rightarrow Executors(at_i) \in Denied\_Users(at_j).
\end{aligned} \tag{5.3}$$

Users that are able to use the permission enabled with authorisation token at are the ones that are authorised to the role in the token, minus the denied users. Users in the set of the denied users might or might not be authorised users of this role.

When a user presents a request to use a permission of an authorised role on some object, access control mechanisms must verify that the permission of the given role is enabled on the requested object, and that this particular user has not been denied the permission.

We introduce a predicate $denied(u, p, obj) : U \times P \times Obj \rightarrow boolean$, which is true when user u is in the set of users denied to use permission p on object obj, otherwise it is false[3]:

$$\begin{aligned}
&denied(u, p, obj) \Rightarrow\\
&\quad \exists t \in T, \exists t\_inst \in TInst(t), \exists at \in AToken(t\_inst) :\\
&\quad p \in PAuth\_Token(at_i) \wedge obj \in OAuth\_Token(at) \wedge\\
&\quad u \in Denied\_Users(at) \wedge CountAuth\_Token(at) \neq 0.
\end{aligned} \tag{5.4}$$

5.4 Implications to the Basic Framework

Context-based authorisations extend the basic framework of role-based access control that was presented in chapter 3. In this section we specify an authorisation state that includes the entities of both role-based and task-related access control. We also redefine access granting rules that now include concepts of enabled permissions and denied users.

5.4.1 Authorisation State

An authorisation state is seen as composed of static elements that are stable except when changes are initiated by administrative actions, and dynamic elements that are tied with activities initiated either by users or as a result of user actions in the system:

[3]A user can be denied to use a permission for other reasons, such as not being authorised for the necessary roles, but this is enforced with other rules in our framework. This particular predicate will be used only to ascertain that the user is not denied to use an enabled permission.

static element $Users, Roles, UserAuthorisationLists,$
$RoleGraph, Tasks, TaskAuthorisationTemplates$

dynamic element $ActiveRoleSets, EnabledPermissions,$
$TaskInstances, AuthorisationTokens.$

Figure 5.4 gives an overview of additional elements and system functions introduced by a task-related framework.

5.4.2 Access Granting Rules

The following access granting rules now apply in addition to the previously defined **Rule 2**.

Rule 2a (Using an Enabled Permission) *A user can use a permission of the user's active role on the specific object only if the permission is enabled for this role.*

$$\begin{aligned}&\forall u \in U, \forall r \in ARS(u), \forall p \in PR(r), \forall obj \in ObP(p)\\&\quad p \in Enabled\_PR(r, obj) \Rightarrow can\_exercise(u, p, obj).\end{aligned} \tag{5.5}$$

Rule 2b (Denial of Enabled Permission) *A user can use a permission enabled for the user's active role on the specific object only if the separation of duty policy is not violated.*

$$\begin{aligned}&\forall u \in U, \forall r \in ARS(u), \forall obj \in Obj, \forall p \in Enabled\_PR(r, obj)\\&\quad denied(u, p, obj) \Rightarrow \neg can\_exercise(u, p, obj).\end{aligned} \tag{5.6}$$

5.5 Discussion - Context Provision

The presented context-based authorisation framework depends largely on the realisation of a task planner. Several established technologies and information systems in use are suitable for this purpose. The basic requirement for such system is control of business activities, that can be mapped to role definitions of our security model. We briefly review two types of such technologies and discuss if they are suitable to our purposes for providing up-to-date information about current activities.

5.5.1 Transactional Workflow Systems

The actions involved in a group of tasks can be viewed as unified to accomplish some business process. For example, the separate tasks for creating order, receiving goods, and paying for the goods could be logically viewed as the process "purchase".

The co-ordinated execution of a group of transactions processing relevant objects by subjects forms an activity that is also referred to as a *transactional workflow*. Many organisations perform the information processing activities using predefined workflows consisting of the numerous steps of transactions with different interactions between them [WH 97]. Transactional workflow models specify, schedule, and execute transactions forming a separate task. They define a task structure and transactional dependencies in it as well as

- Tr = a set of transactions, $tr_1, ..., tr_n$;

  T = a set of tasks, $t_1, ..., t_n$;

  AT = a set of task authorisation templates;

  T_INST = a set of task instances, $t\_inst_1, ..., t\_inst_n$;

  $Auth\_Tokens$ = a set of authorisation tokens, $at_1, ..., at_n$;

  Obj = a set of objects, $obj_1, ..., obj_n$;

  $Type$ = a set of object types, $type_1, ..., type_n$.

- **transactions:** $TrT(t) : T \rightarrow 2^{Tr}$, a function mapping task t to a set of transactions;

- **objects:** $type(obj) : Obj \rightarrow Type$, a function mapping object obj to its type;

  $TypeT(t) : T \rightarrow 2^{Type}$, a function mapping task t to a set of object types that can be processed in it;

- **task instances:** $TInst(t) : T \rightarrow 2^{T\_INST}$, a function mapping task t to a set of task instances;

- **authorisation tokens:** $AToken(t\_inst) : T\_INST \rightarrow 2^{Auth\_Tokens}$, a function mapping task instance t_inst to a set of authorisation tokens;

- **enabled permissions:** $Enabled\_PR(r, obj) : R \times Obj \rightarrow 2^P$, a function mapping role r and object obj to the set of enabled permissions.

Figure 5.4: Additional Elements and System Functions.

the execution requirements[4]. However, the workflow models do not in themselves support authorisations or access control policies[5].

Tasks in Workflows

Tasks in the workflow systems are specified by defining:

- a set of transactions together with an execution structure of each transaction,
- temporal dependencies between transactions in the task,
- task execution requirements.

An execution structure of each transaction is defined by providing execution states and transitions between these states. The execution states of previous transactions may influence the successive order of other transactions in a task. Every transaction can progress through several states, generally defined as *initial, executing, done, on-hold, aborted.* Any of these states of one transaction may be a precondition for the progress of other transactions. An example of such a precondition is "transaction Tr1 can start when transaction Tr2 is done", or "transaction Tr3 can start when transaction Tr2 has aborted".

The processing sequence of the whole task can depend on the output values of its transactions. For example, in the task to approve a sales order, if the credit rating returns a pure outcome for a customer, the sale is not approved and ordered goods are not delivered. This precondition can be stated as "transaction Tr1 can not be started if transaction Tr2 returns a value less than the specified number".

External events, such as expirations and deadlines, may also influence the progress of the task. The examples of such conditions could be "transaction Tr1 can not be started after 8 PM", or "transaction Tr2 must be done within 24 hours, otherwise it is aborted".

Task Scheduler

The evaluation of the defined inter-transaction dependencies and processing of the whole task in the workflow system is performed by a task scheduler. A scheduler is a program that submits transactions for execution and ensures that a transaction reaches an acceptable termination state. It then schedules successive transactions according to the dependencies in each task. A scheduler is also responsible for the whole task reaching an acceptable termination state in an optimal way. It is capable of identifying various alternative ways to proceed with the task and of handling failures.

From this overview it is possible to perceive that the workflow management systems would be very suitable for providing information required by our context-based authorisations. The design could utilise the known task structures, co-ordination requirements

[4]Further reading about transactional workflow systems can be found in Rusinkiewicz [RS 95], Georgakopoulos [GH 95].

[5]Recently there have been several attempts to define authorisation models for workflows(Atluri [AH 96], Bertino [Ber 97, Ber 99]).

of a collection of transactions, and execution semantics of the workflow systems. The workflow task scheduler can be used to give information about terminated and scheduled transactions to enable just-in-time permissions of roles.

5.5.2 Auditing Systems

A journal kept for audit purposes can also be used to identify previous context. If audit history is made available to the access control software, future access request will be conditional on past actions recorded in the journal. However, the audit journal is a very large and complex data structure. The only information that is required in our framework is what transaction was invoked by which user, and in what role it has been used. Further work is needed to standardise content and organisation of audit journals so that suitable context can be developed.

5.6 Conclusions

Though the need for context-based access control has been recognised for some time, there has been little research into specification and enforcement of authorisations from a task-based perspective.

In this chapter we have described an “active” context-based approach to access control. This approach differs from a “passive” subject-object view in many respects. Permissions are controlled and managed in such a way that they are turned-on only if they are to be used in tasks. To enable this, we have defined an authorisation token that indicates what permission of which role is enabled.

Although a number of workflow applications are presently used to control and execute related groups of transactions, they do not in themselves handle permission authorisations, especially in a unified framework with role-based access control. The mechanisms introduced in this chapter could also be useful for authorisation management in the workflow systems. These issues may require further investigation.

Chapter 6

Design and Implementation

6.1 Introduction

In this chapter we will describe how we propose to apply the presented security model with its security constraints and context-based authorisations in computer systems. The RCBS model is independent of the environment in which it may be implemented. For example, the model can be embedded in operating systems or database systems, or implemented at the application level. The implementers are able to choose, based on requirements, between several alternative design approaches.

6.2 Authorisation Database

Access control mechanisms in the RCBS model require that security attributes are kept about users, roles, transactions, objects and tasks, as is illustrated in figure 6.1.

User security attributes describe what roles are authorised to the user.

Role security attributes consist of permissions associated with the role, users authorised for the role, and names of roles that are strongly/weakly exclusive with the role. A role hierarchy describing inheritance relationships between roles may exist as a separate structure.

Types of objects that a transaction can operate on, and the names of tasks in which it can be used are referred to by transaction security attributes.

Object security attributes define the permissions required to operate on the object. According to the classification of objects taken in chapter 4, the object security attributes describe in addition the domain and the conflict of interest class to which the object belongs.

Task security attributes are represented as task authorisation templates, describing permissions that are enabled for each task's instance together with the names of roles to which permissions belong. A defined critical combination of transactions within the task is also a security attribute of the task.

To help with the maintenance of the database of security attributes security packages often support implementation of description files. A description file allows a designer to capture semantic information pertaining to access control requirements of a particular entity. The database may be established by defining and managing description files to represent and capture the essential features. The technical skills required for creation

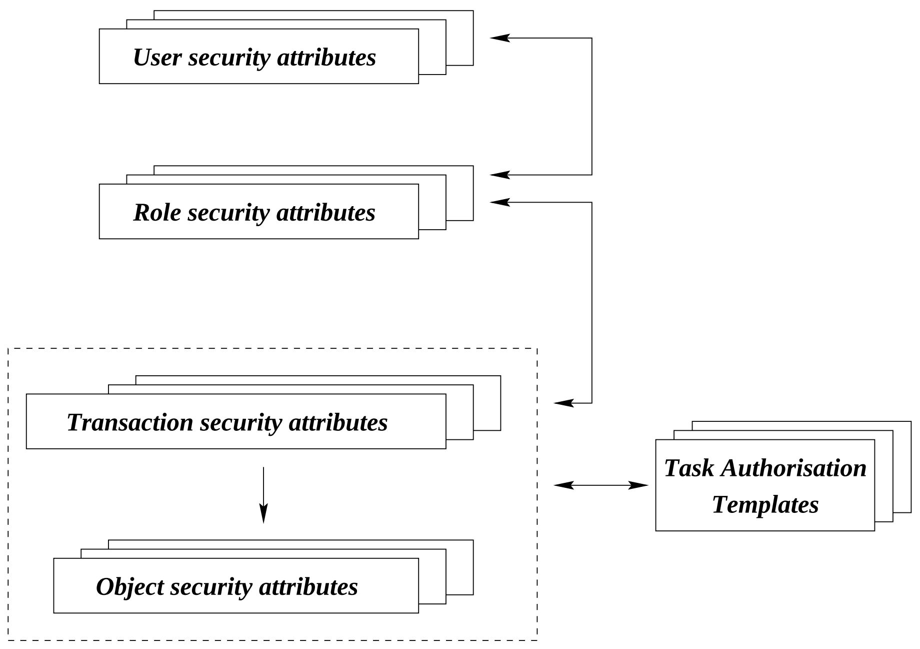
User security attributes
Role security attributes
Transaction security attributes
Object security attributes
Task Authorisation Templates

Figure 6.1: Database of Security Attributes.

and administration of the separate description files differ, decreasing from lower level to highest level. This provides for possible decentralisation of management of different components of the model.

6.3 Design Architecture

In this section we propose a design architecture for the RCBS model. We introduce the components, through which the consistent management of security attributes occurs.

Figure 6.2 gives an overview of all components that enable effective enforcement of security constraints and authorisations defined in previous chapters.

The consistency of the *static* authorisation state is ensured with the help of two main components, called a role manager and a task manager.

A ***role manager*** has to accomplish the following tasks:

1. verify that roles are correctly defined;
2. ensure consistency in role hierarchy;
3. monitor user authorisations to roles.

First task is concerned with the application of constraints that directly deal with permissions given to roles. In addition to ensuring that each role has minimum number of permissions required, the role manager must verify that no role violates an enforced separation of duty policy. This requires applying **Constraint 3** (per-role operational SoD), and **Constraint 5** (per-role object based SoD).

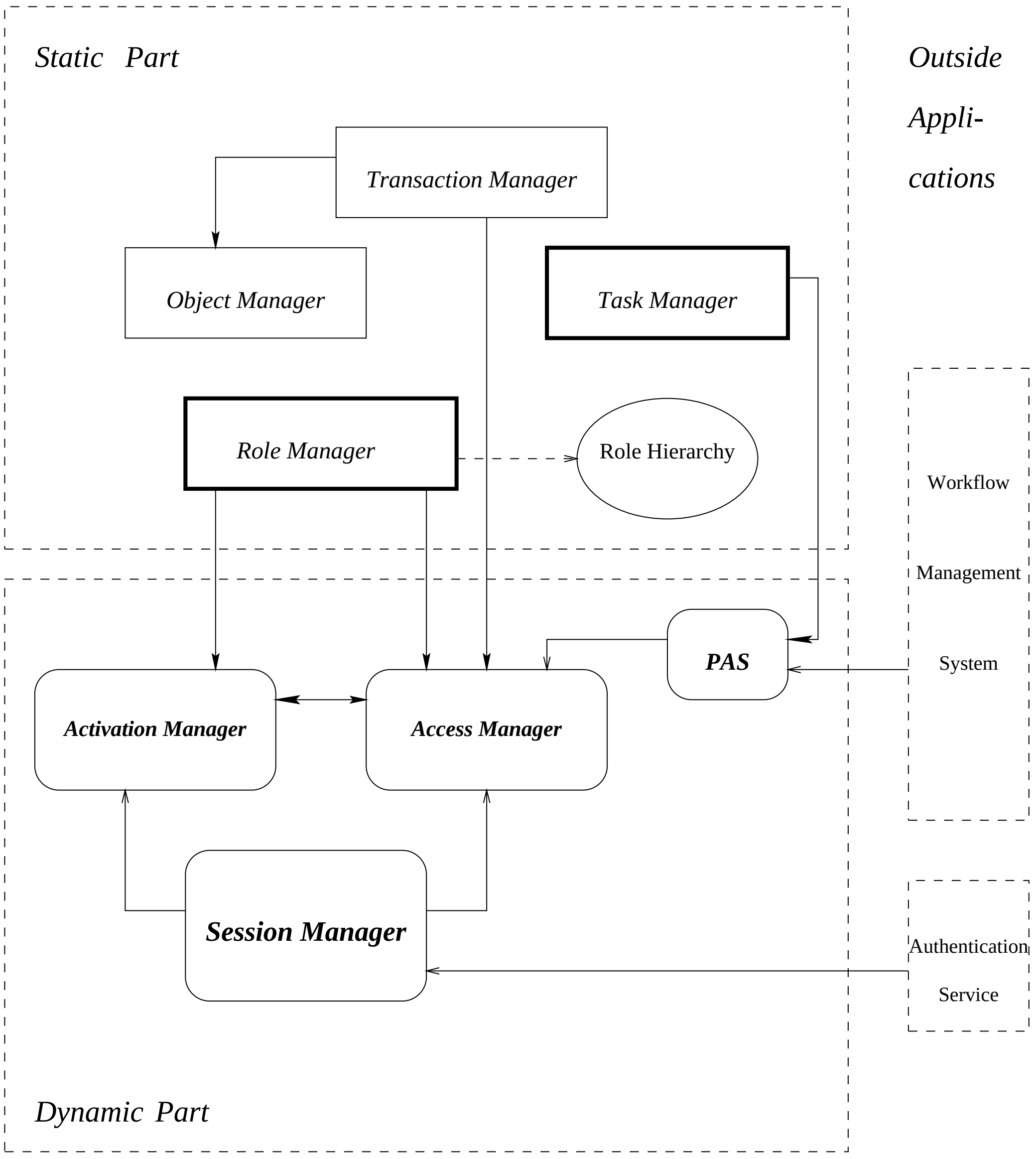

Figure 6.2: The Design Architecture.

Second task ensures consistency of role relationships in role hierarchy according to the properties of SoD policies. Specifically, statements defined throughout **Theorems 1-4** must hold true at all times.

Finally, the role manager has to ensure that users are authorised to the defined roles in consistence with authorisation-time exclusion (**Constraints 1, 1a, 1b**), as well as with operational and object-based SoD properties (**Constraints 4, 6**).

The role manager must also support the main mapping functions. It must provide answers to the queries such as the permissions associated with a role, the transaction and objects included in those permissions, or the set of mutually exclusive role pairs.

A ***task manager*** is required for the enforcement of context-based authorisations. The task manager has to provide information as to what permissions of what roles can be used in a particular task instance. In addition, critical operations of transactions are defined through the task manager.

Other components, such as a ***transaction manager*** and an ***object manager*** are less important in the framework of the RCBS model. A transaction manager has to ensure that transactions operate only on the allowed types of objects, and deals primarily with the internal operating structure of transactions. An object manager, on the other hand, deals with the classification of objects to conflict of interest classes and domains.

The consistency of the *dynamic* state is ensured with the help of the other two components, called an activation manager and an access manager. An ***activation manager*** enforces application of **Rules 1, 1a, 1b** at the role activation time, while the actual access according to the **Rules 2, 2a, 2b** and **Rule 3** is controlled by an ***access manager***. A permission authorisation service (***PAS***) is introduced to provide for the effective enforcement of context-based authorisations at the dynamic level. The functions of these components will be described in more detail later in the chapter when we present processing phases necessary associated the RCBS model.

The security constraints of the RCBS model provide confidence as to adherence of organisational security policies. In theory, similar effects can be achieved through establishment of security procedures among administrators. For example, administrators can maintain and share a list of all known conflicting permissions and ensure that two conflicting permissions are never assigned to the same role. In reality, however, the procedures break down, and administrators get reassigned over time. The constraints, therefore, provide assurance that critical security requirements are uniformly enforced within the system.

6.4 Processing Phases

The different security rules and constraints of the defined security model are applied throughout three processing phases associated with our security model:

Authorisation and Administration This phase consists of creating and maintaining the database of security attributes. Administration tools are usually implemented separately as privileged applications.

Activation The activation phase consists of establishing a session and activating/deactivating roles in a session. A session is a set of processes, acting on behalf of a user. Session establishment involves authenticating the user, creating one or more user processes, and associating user security attributes with each process.

Enforcement In the enforcement phase the access control rules are enforced in order to grant or deny access to objects. User security attributes such as active roles are compared with task requirements and object security attributes to make a final access decision.

In an actual application, a designer may chose between several alternative design approaches in respect as to how and what is processed at each phase. In the next section we describe each phase in more detail and indicate what design decisions should be taken.

6.5 Authorisation and Administration Phase

During authorisation phase, an authorisation database is created, and constraints for the enforcement of high-level security policy are defined. Depending on a particular application, a designer might choose to include constraints pertaining to only one of the several variations of separation of duty presented in chapter 4. For example, application of either history-based SoD or operational SoD may be sufficient in some security policies. A designer is given possibility to chose the SoD policy that is most suitable to the requirements of an application. Once selected, all constraints pertaining to the same SoD policy must be applied.

The RCBS model is treated as enforcing a non-discretionary access control method. That is, one or more administration roles are required to be distinct from other roles. Their permissions must deal only with the policy-defining components of the model, such as user-to-role, role-to-permission, and permission-to-task mappings, and security constraints; users not assigned to administration roles should be denied those permissions.

An example of administration model suitable for managing the components of the RCBS model is presented by Sandhu in [SCFY 97, SBh 97]. The administration tools, should they be implemented, have to communicate with the relevant managers, and by means of fixed operations manage an authorisation database. The integrity is maintained by checking and enforcing security constraints and theorems.

The administration of a role-based security system has to address problematic issues related to the revocation of role authorisations, delegation of role authorisations, and auditing. Since they have been little covered in the existing literature, we feel that is necessary to address those issues in more detail, as far as they apply to our security model.

6.5.1 Revocation of Authorisations

Revocation of a role authorisation from a user involves, in respect to the basic framework of chapter 3, removing the user from the authorisation list associated with the role.

In the RCBS model, if role $R1$ inherits role $R2$, then the permissions of role $R2$ form a subset of permissions of role $r1$. When a user is authorised to role $R1$, this means that all its permissions are available to the user including permissions inherited from role $R2$. When the role authorisation is revoked, all permissions of the role including inherited ones are automatically removed from the user. If the administrator wishes inherited permissions to be left authorised for the user, this operation must be done explicitly by assigning the user to the UAL of the appropriate roles.

The administration tools must constantly remind an administrator about the role hierarchy when a new role is being authorised to the user. This helps to assess that the user has the minimum set of permissions required for his functions.

6.5.2 Delegation of Role Authorisations

A non-discretionary access control method prohibits users not assigned to administration roles from granting role authorisations to other users. However, in case of absence of users authorised to certain roles or inability to use certain permissions from some machines, users may need to temporarily delegate role authorisations to other users. In such cases a possible delegation procedure may be worth considering.

In the RCBS model, a delegation procedure can be introduced by creating a special transaction that operates by temporarily assigning an authorised role to another user. Permission to perform such transaction could be allocated to roles that have to be regularly in use, when absence of authorised users might put activities of others on hold. Temporal role assignment to other users has to be accompanied by some kind of a *delegation token.* This must give particulars as to what role has been delegated, what user originated the delegation, and for what period of time, which could be later analysed for auditing purposes by system administrators.

Delegation, as such, is a complex issue, and many aspects, such as related accountability issues, still require further research before it can be safely introduced to non-discretionary security systems.

6.5.3 Auditing

By their nature, roles provide a simple way to identify what users currently have access to permissions associated with them. The constraints can also provide an information as to how these roles are authorised to users, and how permissions are allocated to roles. In the RCBS model, the points have been indicated where different constraints could be introduced, and the mechanisms have been proposed to enforce them. This provides administrators and auditors with the capability to determine how an active access is granted to users. At activation and enforcement phases, the information is recorded about current role activations and permission usage. This information can be used for auditing to answer queries about activities in the system.

6.6 Activation Phase

In order for a user to activate authorised roles, *a session* must be established. A session is a set of processes acting on behalf of a user. Session establishment involves authenticating the user, creating user processes, and associating user security attributes with these processes. The full activation phase in a computer system consists also of changing the characteristics of, and removing sessions.

The rules under which users activate roles (**Rules 1, 1a, 1b**) are applied during the activation phase. Up-to-date values of user security attributes, such as authorised roles,

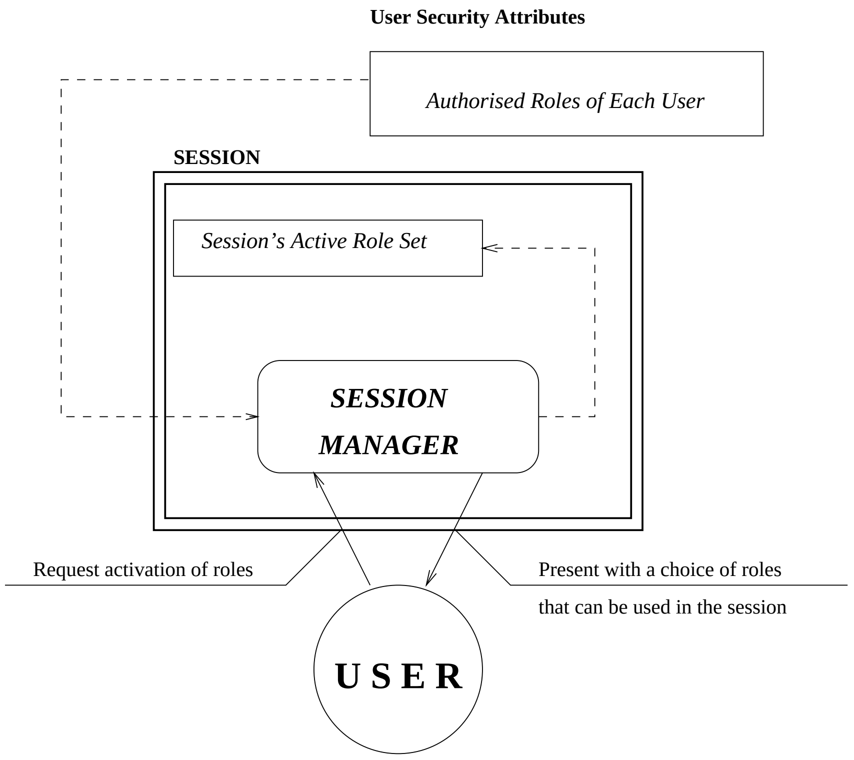
User Security Attributes
Authorised Roles of Each User
SESSION
Session's Active Role Set
SESSION MANAGER
Request activation of roles
Present with a choice of roles that can be used in the session
U S E R

Figure 6.3: Session Manager.

must be available in order for user processes to be able to access roles in the name of the user.

One of the following approaches may be implemented for keeping up-to-date values of user information:

uncached user information User information including the roles authorised for each user is obtained through the role manager;

cached user information User information is kept on each server, and this information is used during the activation phase without having to access role manager.

Keeping an uncached user information guarantees that this information is always up-to-date but requires communication with the role manager whenever a session is established. In addition, failure in communication results in that no sessions can be processed.

Cached user information requires updates whenever there is a change in user information, such as new roles being authorised, but does not require communication during session establishment. Since it is generally assumed, that session phase is more frequent than occurring changes in user information, most implementations use cached user information, so reducing network usage.

6.6.1 Session Manager

We introduce a special component, called a *session manager*, to start processes on behalf of a user in a session. Each established session in a computer system must have a separate session manager.

As is illustrated in figure 6.3, during the activation phase the session manager enables the user to activate/deactivate roles in a session. It also handles access requests on behalf of the user.

The operations that have to be performed by the session manager after a session has been established are:

1. to analyse user security attributes,
2. to identify authorised roles,
3. to apply role activation rules,
4. to present the user with roles that can be used in a session.

6.6.2 Activation of Roles

One of the several options is available for managing activation of roles in a session. One option is to have all roles authorised for a user automatically activated once the session is established. The extreme opposite of this is to restrict a user to only one active role per session. However, in the first case the roles are activated that may never be used by the user, whereas the second case increases the likelihood of users trying to bypass security procedures. The third option is to let users choose the authorised roles they want to activate in each session.

Since weak exclusion constraints (**Rules 1a, 1b**) prohibit simultaneous activation of certain roles, a user will not always be able to activate just any role from the set of authorised roles. Firstly, a session manager must identify the different subsets of roles allowed to be active at the same time. The user can then select to activate one or more roles from one of the identified subsets. The user should also be allowed to deactivate already active roles and to activate new roles in the same session, while activation constraints apply on every role activation.

6.6.3 Multiple Sessions

To give the required flexibility and convenience, users are often allowed to have multiple sessions opened simultaneously in the system. This requires to consider whether each session opened by the same user should have the same or different set of active roles (ARS).

When the roles are not constrained on activation, the situation that each session may be associated with a different ARS is probably acceptable to most security administrators. For example, a user who is involved in several projects benefits from having a window open for one project, and on the same display, a window open for another project.

However, when role activation is constrained with the mutual exclusion among roles, the situation that a user on a workstation screen has one window open with one ARS and another window with a different and conflicting ARS might be not at all acceptable, since it may not be consistent with the enforced separation of duty policy. For example, a bank teller can also be an account holder at the same bank. The policy in a bank is to constrain employees who are also account holders from being active simultaneously in both roles. The situation where an employee has one window open for the bank teller

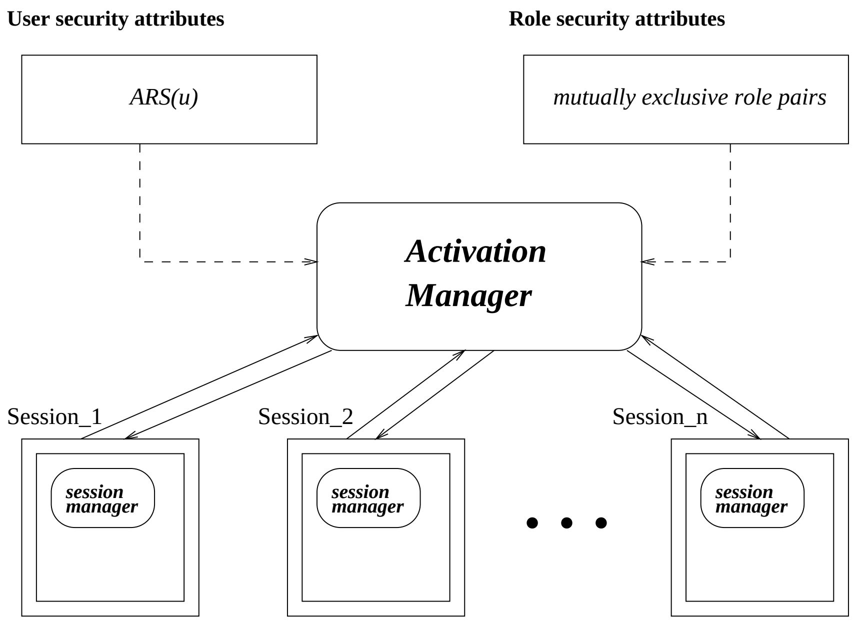

Figure 6.4: Activation Manager.

role and another window on the same display for the management of his account is not consistent with this policy.

One solution is to restrict a user to a single session at a time. Another solution considered by Barkley in [Bar 97] is to associate a single unique ARS with a given number of sessions. When a user changes an ARS in any of these sessions, each ARS of every other session from the identified set is also updated. This approach ensures that a given set of sessions will not have conflicting roles active simultaneously. In large computer systems this solution, however, might seriously impact network throughput, since there will be constant communication among the sessions opened by the same user.

6.6.4 Activation Manager

For controlling multiple sessions in an implementation of the RCBS model, we propose a solution based on a slightly modified variation of the above approach.

We first introduce an additional component, called an *activation manager*, to handle role activation requests passed on by session managers from sessions started in the system[1], as is illustrated in figure 6.4.

For each user a dynamically updated list of currently active roles is kept by the activation manager. The ARS in every session opened by the same user is then allowed to be different. However, the activation manager must be contacted on every role activation/deactivation in a session, so that the main ARS for each user is up-to-date.

We also require that all ARSs kept by the activation manager must always remain internally consistent. New roles can be activated in the established sessions and new sessions can be opened with different ARSs, as long as no conflicts arise among the

[1]In actual implementations, a number of activation managers will probably have to be implemented, each handling requests from sessions started in subsystems with smaller number of users.

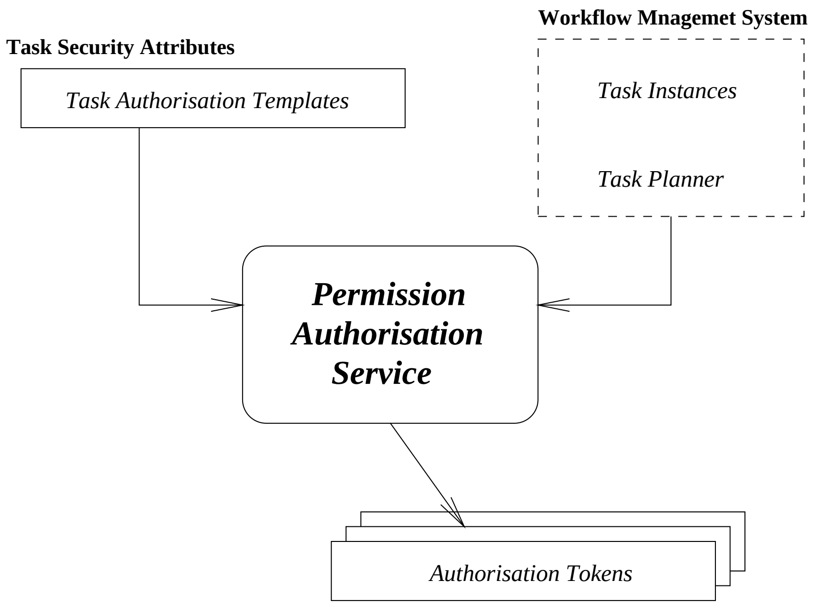

Figure 6.5: Permission Authorisation Service.

already activated roles and the newly requested roles by the same user. This ensures that no mutually exclusive roles are simultaneously activated by any user.

Although this method ensures effective control of multiple sessions, the activity in a network of computers is again increased. A number of sessions opened by the same user could be limited in order to reduce the number of required communications. However, no one solution will satisfy all needs in this situation. For each specific application a designer must choose a level of (in)convenience that is appropriate for the current environment, its risk, and level of user discipline.

6.7 Enforcement Phase

The enforcement phase consists of analysing permissions of activated roles in order to grant or deny access to transactions that operate on objects. This phase occurs every time a user, i.e. a process acting on behalf of a user, attempts to perform transactions in order to manipulate the application's objects. In a network environment, an implementation must ensure that the enforcement phase has up-to-date information about the permissions of activated roles.

In the proposed design architecture of the RCBS model, the role manager has access to role hierarchy, and can give information about current effective permissions of a role. The information as to which of these effective permissions can actually be used at the specific moment must be available from a different service.

6.7.1 Permission Authorisation Service

We propose implementing a *permission authorisation service* (PAS) for the enforcement of context-based authorisations. PAS has to register what tasks have been started in the system, and accordingly enable/disable permissions of relevant roles. Figure 6.5 illustrates the basic operations that have to be performed by PAS. They are the following:

1. to contact the task planner about the completed transactions and the transactions scheduled for possible execution;
2. to generate an authorisation token for the permission that allows to execute the scheduled transaction;
3. to keep track of permissions used; each time an enabled permission has been used the associated usage count must be reduced;
4. to record users that use the enabled permissions.

In this approach, we assume that a task planner is available from a workflow management system. It schedules transactions according to the plan defined in the definition of a task. PAS enables permissions to scheduled transactions, taking information from the task authorisation templates. In principal, PAS cannot enable a permission to perform a transaction, if this transaction has not been scheduled.

An authorisation token for enabled permissions is available from PAS. When a user requests access to a certain transaction, PAS is contacted. If there exists an authorisation token that allows to use the requested transaction on the indicated object for a user active in the correct role, and if the usage count in this token has not been exceeded, the request is granted. If otherwise, the request is rejected.

6.7.2 Access Manager

To evaluate access requests and to enforce access decisions according to the defined **Rules 2, 2a, 2b**, and **Rule 3**, we introduced yet another component, called an *access manager*.

Figure 6.2 shows that to reach an access control decision in respect to the rules introduced in the RCBS model, an access manager must have information from the role manager about the effective permissions associated with active roles, from PAS about which of the effective permissions are enabled, and from the transaction manager about the types of objects that can be accessed.

6.8 Conclusions

The design decisions that can be taken towards the implementation of the RCBS model are varied and are mostly dependent on requirements of a specific application. In this chapter we have emphasised the points where it is necessary to take one of several alternative approaches.

The functionality of the RCBS model depends on successful implementation associated with each of processing phases, including set-up of the main database of access control information, establishment and management of sessions, enforcement of access control, and overall maintenance with the help of administration tools. The RCBS model has been formulated at the level of abstraction that is intuitive and consistent with the way duties and responsibilities are shared and business is conducted in the commercial or government organisation, making security administration easier. Enforcement of security constraints gives additional assurance as to adherence to well-known security principles and policies that are important in business practices.

Chapter 7

Authentication

7.1 Introduction

Applying the presented RCBS model to distributed computing environments requires to reappraise the established security framework. Each service in the model could be implemented on distributed hosts that make their assertions without reference to a mediator, such as a centralised resource that keeps an authorisation database. The mechanisms to enable this must be dynamic and easily used by all services while maintaining strong assurance. Specifically, the hosts should be able to collect and authenticate the necessary assertions (identity, roles, use-conditions), based on which a specific service will be granted.

In this chapter we look at one of the fundamentals concerns in building a secure distributed system, *authentication*, and how it affects the design of our model. Specifically, we examine two existing approaches to authentication in distributed systems, namely ticket-based approach enforced in Kerberos, and certificate-based approach used in SPX and PKI. We look at how effectively each of these approaches can be combined with the RCBS model. We conclude by presenting an authentication framework that to be used in applying the RCBS model to distributed computing environments.

7.2 Authentication in Distributed Systems

A distributed system presents some unique security problems. A fundamental concern is authentication of various entities in the system. In a distributed system, entities communicate by sending and receiving messages over the network. Various services are distributed among different servers across the network. The processes acting on behalf of users direct service requests to the appropriate servers.

An assured authentication framework is vital for servers to be able to identify and verify an identity of the user making a request. Only after successful authentication the server can proceed with access control decision to its service.

All authentication procedures involve checking known information about a claimed identity against an information supplied by the claimant. In the very primitive authentication measures, users are required to enter their passwords in order to access a server. Such measures, however, are seriously inadequate in distributed systems. A detailed survey of all authentication aspects in distributed systems is presented by Woo and Lam [WL 97].

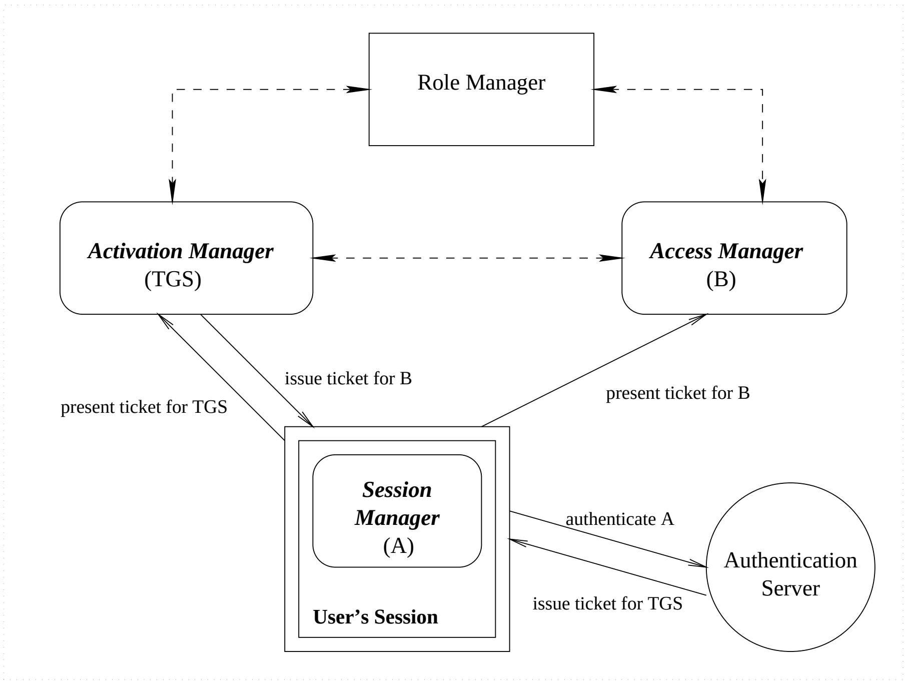

Figure 7.1: Ticket-based Authentication.

In the next sections we examine two of the most popular approaches to authentication, and present how they can be applied in the design of the RCBS model.

7.3 Ticket-based Approach

A centrally enforced, symmetric cryptography based authentication scheme is defined in the Kerberos model, the overview of which is presented by Neuman and Ts'o in [NT 94]. Kerberos provides secure network authentication, allowing a user to access different servers on the network. Authentication is built around the concept of *tickets*. A ticket is used to pass securely to the server an identity of the user for whom the ticket was issued. The user has to obtain tickets for every service he wants to use on the network.

The authentication protocol in this architecture is quite straightforward. A user first requests an initial ticket for ticket granting service (TGS) from a Kerberos authentication server (KAS). This ticket is sent to the user, encrypted with a secret key shared between the user and KAS. Using this ticket, the user can request other tickets for services on the network from TGS. The appropriate tickets are then presented to the servers providing the required services, along with the user's authenticator. Based on the outcome of the authentication protocol, a server lets/denies access to its service.

Kerberos authentication scheme could be used to provide assured authentication between different services of the RCBS model. In a network of computers, services, such as a role manager, activation manager, and an access manager can be distributed over a number of different servers. A user logging into the computer system will need, after initial authentication by KAS, to access both an activation manager and an access manager.

Figure 7.1 indicates the communication steps that have to take place between different services for a user to access resources in the RCBS system. A central authentication server first authenticates the user. It issues an initial ticket to access the activation manager. The activation manager can act as a ticket granting service in this case, since, according to our security policy, a user may access other services only after the necessary roles have been activated. Alternatively, a separate ticket granting server could be implemented that issues tickets to all services in the system.

The activation manager contacts the role manager to verify that the requested roles are authorised to the user. Having activated the roles, the activation manager supplies the user with tickets to access other services in the system, including the access manager.

There are, however, several limitations associated with this authentication scheme. Kerberos enforces primarily a centralised security model with the authentication server at the heart of it, and although connections between different administrative domains can be established with the help of Kerberos realms, this leads to an increased complexity of the scheme. Another limitation is that it requires servers to be online. The KAS has to be on-line during login, and the activation manager is needed when tickets are requested. In addition, the role manager has to be contacted on every role activation.

7.4 Certificate-based Approach

We will now consider another approach to authentication on distributed systems, built around the digitally signed statements, called *certificates*.

In the existing certificate-based authentication schemes, such as SPX [TA 91], or more recent PKI [FHPS 99], a certificate refers to digital binding of a principal's[1] name to its long-term public key. This certificate, called ***public key/identity certificate***, is used for identity authentication, that in addition requires proving that the principal knows the associated private key. Identity certificates are issued by *certification authorities* (CA). They verify the connection between an identity of a principal and possession of a public key/private key pair before signing the certificates.

Lately, there is a tendency to use certificates for attesting also other information, such as various attributes associated with a principal [FH 99] or use-condition information [TJM 99]. For example, when a party wants to authenticate a principal authorised for some role, this means that the party requires assurance not only as to an identity belonging to a claimant, but also that at the time of a request the claimant is authorised to a certain role.

We may recall that previously, in order to ascertain a user's authorisation to a role, the service had each time to access a role authorisation database held by the role manager. Certificate-based approach, however, allows pieces of an access control policy to be stored in distributed certificates. Names of the roles authorised to a user can then be recorded in digitally signed certificates to be exchange between services on the network during mutual authentication.

Components that enable certificate-based authentication include reliable mechanisms for generating, distributing and verifying certificates. All these mechanisms rely on public-key cryptography for digital signatures, a public key infrastructure for certificate management, and protocol for secure communication.

[1]An entity that constitutes a principal can be a user, a host, or a process acting on behalf of a user.

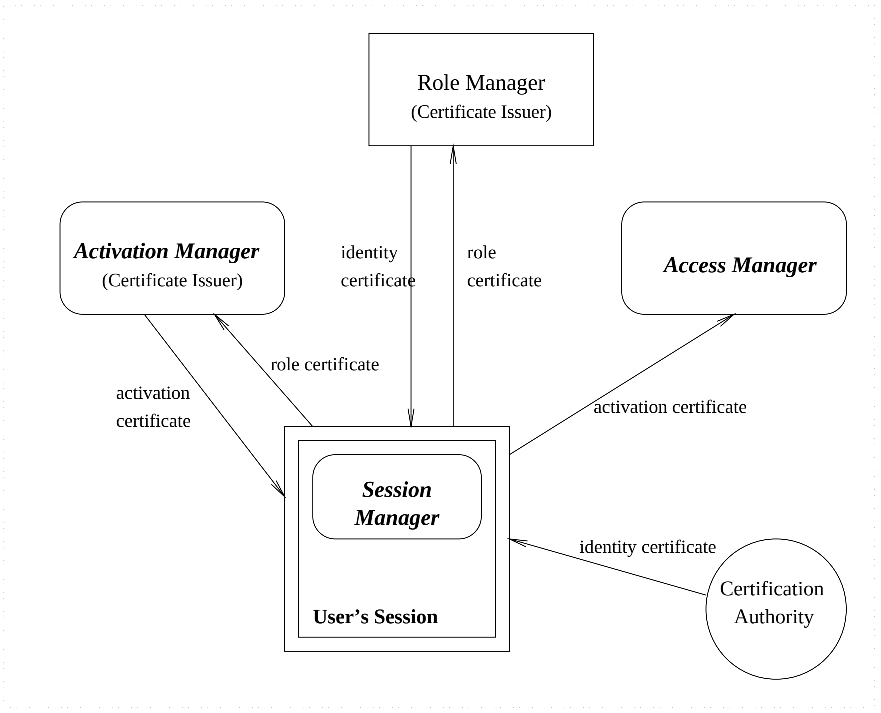

Figure 7.2: Certificate-based Authentication.

Advantages of the certificate-based approach over the established Kerberos systems are decentralisation and reduced network activity since servers can be off-line. However, this approach also requires that users and some of the services should get hold of a public key/private key pair, and use more expensive cryptographic techniques. Final decision depends on the requirements of a particular application. It is our expectation that PKI identity certificates will become more widespread than Kerberos identities since many organisations are beginning to use them for employee identification. For this reason we present in the next sections a detailed architecture for deployment of certificate-based authentication in the RCBS system.

7.5 Authentication Architecture

Figure 7.2 shows an overview of authentication architecture in our security model indicating only the major components. The heart of the system is the *session manager* where all the user's requests originate. The *session manager* first requests on behalf of the user a ***role certificate***, presenting an identity certificate during communication with the role manager. The *role manager* verifies the identity certificate, and then from its database of UALs generates a list of roles authorised to the authenticated user. A digitally signed role certificate is sent to the *session manager*. This certificate can be cached at the local machine. Since it will not be necessary to obtain another role certificate until the previous one expires[2], the *role manager* can be off-line.

[2]Exception is the case of some of the authorised roles being revoked from the user. Then the old certificate has to be revoked, and a new one issued.

One of the functions of the *session manager* is to activate the requested roles on behalf of a user. For this purpose the *activation manager* is contacted. In the course of mutual authentication, the *session manager* sends the cached role certificate together with the names of requested roles. After identifying the user and verifying the role certificate, the *activation manager* either activates or refuses to activate the roles. A digitally signed ***activation certificate*** is issued as a proof of successful activation. It will be valid for the current session.

Having activated the roles, the user can proceed to execute transactions and request access to recourses. In this case the *access manager* is contacted. The cached activation certificate is presented during mutual authentication between the session manager acting on behalf of the user and the access manager. The access manager then acts according to the access granting rules to allow/disallow requested services.

7.6 Generating and Managing Certificates

In the described framework three types of certificates are being used: user identity certificates, role certificates, and activation certificates. We assume that the identity certificates are generated and managed by certification authorities (CAs) and certificate distribution centres of the public key infrastructure, that typically also supports some form of certificate revocation.

7.6.1 Format

Role and activation certificates are the declaration statements.

A ***role certificate*** declares an identity of a user and names of the roles authorised for this user.

An ***activation certificate*** binds the user's identity with the names of activated roles and an id of the current session.

These certificates do not, in themselves, specify any associated public key, or define how the roles were entered or how the certificate must be used. Additional values include a validity period and a unique ID for the certificate.

Both the role manager and the activation manager generate the appropriate certificates according to the access control information held in the system, based on having successfully identified the user. The result is digitally signed with the certificate issuer's private key. The receiver of those certificates will need to obtain corresponding public keys separately, in order to check the signature on each certificate.

7.6.2 Life-Time

A lifetime of a certificate depends on how long-lived is the declared information. For example, identity certificates containing the name and long-term public key usually have validity periods of months or even years.

Role certificates used in our system bind a user's identity with the names of authorised roles. Assuming that job related tasks and responsibilities assigned to a person are relatively stable, role authorisations will not change very often. Therefore, the validity of role certificates could be almost as long as of identity certificates. This inevitably means that some form of revocation mechanism must exists, in case role authorisations are removed from a user before the certificate expires.

A certificate can be revoked by maintaining a certificate revocation list or by deleting certificates from the role manager's database. This requires for the role manager to be on-line during verification of certificates. In an actual implementation, a certificate revocation list could be available from a different server.

Activation certificates declaring that a user has activated certain roles have a lifetime spanning only one session. We may safely say that they will expire by the end of working hours. Having such short-lived certificates has advantage, because they will rarely need to be revoked before expiring. Some of the features of short-lived certificates are presented by Hsu and Seymor in [HS 98].

7.6.3 Distribution

Different approaches for obtaining the digitally signed certificates can be taken. Each approach could be made to work within our model, and we will now compare their relative advantages and disadvantages.

User-Pull Architecture

One method is a user-pull scheme, where a user or, more accurately, the process acting on behalf of the user in a session pulls the appropriate certificates from the certificate issuers and then presents them to other servers.

In describing an authentication framework above, we have assumed this particular approach. A user, Alice, first needs to download her role certificate issued by the role manager and store it in her machine. Later, when Alice activates roles this certificate is presented by her machine to the activation manager. The activation manager, in turn, issues to Alice the activation certificates, that are stored in her machine, and are later presented to the access manager. Each service uses the information retrieved from the certificates for its own purposes, e.g. for activation of roles or for granting access to certain objects.

The user-pull method supports high user mobility. A user can download the necessary certificates to her current machine. The user's machine can then present those certificates to remote servers along with her identity information.

Server-Pull Architecture

Another method is a server-pull scheme, where each server pulls the appropriate certificates from the certificate issuers as needed, and then uses them for its own purposes. In this architecture, a user, Alice, does not need to download her role and activation certificates. Instead, she needs only to prove her identity.

The remote server first authenticates the user. The server then retrieves the certificates required for supplied services from the appropriate certificate issuers, e.g. the activation manager gets the role certificate from the role manager, and the access manager obtains the activation certificate from the activation manager.

In this architecture the certificate obtaining mechanism is transparent to a user, while limiting user portability.

7.7 Authentication Across Domains

Up to now we have assumed a centralised certification authority and centralised activation and access managers. A realistic distributed system, however, often interconnects several independently administered subsystems. The emergence of Internet also increased an amount of co-operative work between people from different organisational domains separated by large distances.

Interaction between principals from different subsystems depends largely on inter-domain authentication. The defined framework is sufficient for identity and role authentication within the local system, since services and a certification authority can easily verify certificates issued locally. However, a server has no way of verifying a request from a remote principal, even if the request is certified by some remote certification authority.

For inter-domain authentication, two issues need to be addressed: *naming* and *trust*.

Naming is concerned with ensuring that principals are uniquely identifiable across domains. Since existing authentication schemes are mostly based upon identification of a name or identity of a principal, they are not sufficient for large-scale systems. One of the emerging solutions is to extend an identity information with additional attributes, as presented by Sandhu *et al* [SP 99]. An attribute may be some particular property of a principal helping to uniquely identify the principal in an outside computer system. For example, authorisation to some role could imply that certain trust can be placed upon a user, or that the user is the one to initiate a particular action.

An access control policy in the RCBS model prohibits any user from activating roles and accessing services, if the user's id has not been found in the local user-role authorisation database. If, however, the role certificate is used during inter-domain authentication, the server has in addition to the user's id a certified information about the roles that are authorised to the user on some remote system. Based on this additional information the server might be able to proceed by allowing activation of local roles. This requires that the server should have a mapping mechanism that maps the names of remote roles declared in role certificates to the local role names. Besides that, the server must also be willing to trust a certification made by a remote authority regarding a remote user. Such trust relationships must be explicitly established between domains.

The success of extending the RCBS model to meet the requirements of enterprise-wide security in large-scale computer systems, therefore, depends largely on further research in inter-related areas.

7.8 Conclusions

In this chapter we looked at implications of applying the presented RCBS model to distributed environments. Authentication is a fundamental concern in distributed systems, that also has a serious impact on the security of our model. We examined existing authentication schemes, and presented a framework that can be used with the security mechanisms of the RCBS model.

Chapter 8

Conclusions

This dissertation presents a security model for access control and authorisation management that allows fine grain specification and enforcement of enterprise-specific policies and internal controls. This chapter summarises the main conclusions and suggests further work.

8.1 Summary

Two fundamental problems that this work attacks are the deficiency of formal expressions for capturing security policies of separation of duty and Chinese Walls, and the lack of mechanisms for the active runtime management of authorisations and related access controls. The main goals are achieved by developing properties of role-based access control, and integrating them with context-based authorisations. The essential features, arising from these domains, may be summarised as follows.

- The application of security constraints, especially for expressing separation of duty policies.

  Separation of duty is a well-known principle of computer security. However, it is difficult to express and effectively enforce this principle in security systems. Although different variations of separation of duty exist, there are few mechanisms necessary to uniformly support this variety of separation of duty policies in a security model.

  In chapter 4 a framework of security constraints was presented that enables application of different variations of separation of duty policy at the role-based access control level. The constraints deal with user assignment to roles, and with permission distribution among roles, including role hierarchies. Chapter 5 developed this further by presenting mechanisms for runtime application of separation of duty policy. In chapter 6 multiple sessions are constrained when separation of duty constraints are in effect.

- The enforcement of authorisations from the perspective of activities in the tasks.

  In the traditional security models[1], access and authorisation control is based on the statically defined relationships between subjects and objects. If a subject requests

[1]A role-based approach that does not consider role or transaction activations also falls into this category.

access to an object, and the subject has relevant rights, access is granted without any further controls. However, in such view access requests are separated from larger context, such as activities of other subjects, history of operations on objects, and current state of business tasks in an organisation.

The abstractions and mechanisms defined in chapter 6 provide for the active management of authorisations as tasks progress to completion. Permissions of roles are constantly monitored, and are enabled and disabled in accordance with context emerging from progressing tasks. Such task-based approach represents radical departure from traditional passive security models.

The proposed mechanisms can be effectively layered on top of role-based controls. The points were also indicated where the workflow or other sequencing technologies might be added.

The trade-offs of the RCBS model are made on the grounds of multi-point protection and flexibility:

Multi-point protection Most of the traditional approaches for controlling a subject's access to objects introduced control only at the point of individual objects. Specification of access control lists (ACLs), for example, is the most popular way for defining subjects' rights to objects. A subject is then granted an access if an appropriate access right to an object is specified in the ACL.

In the RCBS model, authorisations are embodied through several components, such as permissions assigned to roles, role inheritance relations, task authorisation templates, and, finally, user-role authorisation lists. These components collectively determine what a particular user is allowed to access at a certain moment in the system. In addition, an access control policy can be enforced not only at the static level, but also at the time authorised roles are being activated by users.

Flexibility The presented security model is not oriented towards dome specific organisational or computing environment. The design architecture presented in chapters 6 and 7 provides only the guidelines for applying the model in computer systems. The granularity of roles and permissions can be specified to suit a specific environment. The designer is able to choose between several alternative approaches for the enforcement of separation of duty policies and other security constraints. The mechanisms introduced for the active runtime management of security provide possibilities for the administrators to define security policies with fine granularity. Because the security policies can, and usually does, change over the system life cycle, the RCBS model offers a key benefit through its ability to meet changing organisational needs.

8.2 Further Work

Further work can exploit mainly the areas of policy extensions to specify inter-organisational issues, to integrate with workflow management systems, and to reason about auditing and accounting. Formal analysis may also require further work.

Inter-organisational issues Policy extensions are required for application of the RCBS model over multiple organisational subsystems. Of particular importance are facilities that allow interaction between components from different administrative domains.

Integration with Workflow Systems For context-based authorisation controls of the RCBS model to be effective, we have to seek ways where they can be automatically enforced at runtime when the corresponding tasks are invoked. In recent years, workflow management systems have gained popularity in commercial sectors. They are used to coordinate and streamline business tasks in an organisation. Integrating context-based authorisations with workflow systems is a broad objective for future research efforts.

Auditing Many mechanisms presented in the model are designed to record current and previous activities in the system. As such, they ought to prove useful when considering the issues of auditing and accounting.

Formal reasoning The formal specifications presented in the model mainly provide ground to reason about the influence of constraints on the access decisions. The ability to reason about the effect of changes in role hierarchies requires further work.

Bibliography

[AH 96] V. Atluri, W-K. Huang. An Authorisation Model for Workflows. In *Lecture Notes in Computer Science*, ESORICS'96, pages 44-64, 1996.

[Bad 89] Lee Badger. A Model for Specifying Multi-Granularity Integrity Policies. In *Proceedings of IEEE Symposium on Security and Privacy*, 1989.

[Bal 90] Robert W. Baldwin. Naming and Grouping Privileges to Simplify Security Management in Large Databases. In *Proceedings of IEEE Symposium on Security and Privacy*, 1990.

[Bar 97] John Barkley. Comparing Simple Role-Based Access Control Models with Access Control Lists. In *Proceedings of 2nd ACM Workshop on Role-Based Access Control*, 1997.

[Ber 97] E. Bertino, E. Ferrari, V. Atluri. A Flexible Model Supporting the Specification and Enforcement of Role-Based Authorisations in Workflow Management Systems. In *Proceedings of 2nd ACM Workshop on Role-Based Access Control*, 1997.

[Ber 99] E. Bertino *et al.* The Specification and Enforcement of Authorisation Constraints. In *ACM Transactions on Information and System Security*, Vol.2 No.1, February 1999.

[BN 89] David F.C. Brewer, Michael J. Nash. The Chinese Wall Security Policy. In *Proceedings of IEEE Symposium on Security and Privacy*, 1989.

[CSRB 92] *Computer Security Reference Book.* K.M. Jackson, J. Hruska, D.B. Parker (eds.). Butterworth-Heinemann Ltd., 1992.

[CW 87] David D. Clark, David. R. Wilson. A Comparison of Commercial and Military Computer Security Policies. In *Proceedings of IEEE Symposium on Security and Privacy*, 1987.

[DMc 89] John E. Dobson, John A. McDermid. A Framework for Expressing Models of Security Policy. In *Proceedings of IEEE Symposium on Security and Privacy*, 1989.

[DTH 92] S.A. Demurjian, T.C. Ting, M.-Y. Hu. Requirements, Capabilities, and Functionalities of User-Role Based Security for an Object-Oriented Design Model. In *Database Security V: Status and Prospects*, 1992.

[DTH 94] S.A. Demurjian, T.C. Ting, M.-Y. Hu. Unifying Structural and Security Modelling and Analyses in the ADAM Object-Oriented Design Environment. In *Database Security VIII: Status and Prospects*, 1994.

[DTHD 96] S.A. Demurjian, T.C. Ting, M.-Y. Hu, T.A. Daggett. User-Role Based Security (URBS) Enforcement Mechanisms for Object-Oriented Systems. In *Database Security IX: Status and Prospects*, 1996.

[FB 97] David F. Ferraiolo, J. Barkley. Specifying and Managing Role-Based Access Control within a Corporate Intranet. In *Proceedings of 2nd ACM Workshop on Role-Based Access Control*, 1997.

[FBK 98] David F. Ferraiolo, J. Barkley, D. Richard Kuhn. A Role Based Access Control Model and Reference Implementation within a Corporate Intranet. In *Proceedings of 3rd ACM Workshop on Role-Based Access Control*, 1998.

[FGL 93] D.F. Ferraiolo, D.M. Gilbert, N. Lynch. An Examination Of Federal and Commercial Access Control Policy Needs. In *Proceedings of 16th National Computer Security Conference*, 1993.

[FH 99] S. Farrell, R. Housley. *An Internet Attribute Certificate Profile for Authorisation.* Internet Engineering Task Force, an Internet Draft. www.ietf.org.

[FHPS 99] W. Ford, R. Housley, W. Polk, D. Solo. *Internet X.509 Public Key Infrastructure.* Internet Engineering Task Force, an Internet Draft. www.ietf.org.

[FK 92] David F. Ferraiolo, D. Richard Kuhn. Role-Based Access Controls. In *Proceedings of 15th National Computer Security Conference*, October 1992.

[FKC 95] David F. Ferraiolo, D. Richard Kuhn, J.A. Cugini. Role Based Access Control: Features and Motivations. In *Proceedings of Computer Security Applications Conference*, December 1995.

[GI 95] Luigi Giuri, Pietro Iglio. A Formal Model For Role-Based Access Control With Constraints. In *Proceedings of 9th IEEE Computer Security Foundations Workshop*, 1996.

[GGF 98] Virgil D. Gligor, S I. Gavrila, David Ferraiolo. On the Formal Foundations of Separation-of-Duty Policies and their Composition. In *Proceedings of IEEE Symposium on Security and Privacy*, May 1998.

[GH 95] D. Georgakopoulos, M. Hornick. An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure. In *Distributed and Parallel Databases*, pages 119-153, 1995.

[GL 95] Virgil D. Gligor, Sven Lorenz. Role-Based Security Administration. Draft 10/13/1995.

[HS 98] Yung-Kao Hsu and Stephen P. Seymour. An internet security framework based on short-lived certificates. In *IEEE Internet Computing*, March/April 1998.

[HT 95] R. Holbein, S. Teufel. A Context Authentication Service for Role-Based Access Control in Distributed Systems - CARDS. In *Proceedings of IFIP 11th International Conference on Information Security*, 1995.

[HTB 96] R. Holbein, S. Teufel, K. Bauknecht. The Use of Business Process Models for Security Design in Organisations. In *Proceedings of IFIP 12th International Conference on Information Security*, 1996.

[Jan 98] W. A. Jansen. Inheritance properties of role hierarchies. In *Proceedings of 21st NIST-NCSC National Computer Security Conference*, October 1998.

[Kon 95] W. Fred. de Koning. Security Within Financial Information Systems, as seen from an auditor's point of view. In *Proceedings of IFIP 11th International Conference on Information Security*, pages 26-38, 1995.

[Kuhn 97] D. Richard Kuhn. Mutual Exclusion of Roles as a Means of Implementing Separation of Duty in Role-Based Access Control Systems. In *Proceedings of 2nd ACM Workshop on Role-Based Access Control*, 1997.

[LABW 92] B. Lampson, M. Abadi, M. Burrows, E. Wobber. Authentication in Distributed Systems: Theory and Practice. *ACM Transactions on Computer Systems*, Vol. 10, No. 4, 1992.

[LW 91] L.J. La Padula, J.G. Williams. Toward a Universal Integrity Model. In *IEEE Computer Security Foundations Workshop*, June 1991.

[MD 94] I. Mohammed, David. M. Dilts. Design for Dynamic User-Role-Based Security. *Computers & Security*, 13 (1994).

[ML 93] *Many-Sorted Logic and Its Applications.* K. Meinke, J. V. Tucker (eds.). John Wiley & Sons Inc., 1993.

[NP 90] Michael J. Nash, Keith R. Poland. Some Conundrums Concerning Separation Of Duty. In *Proceedings of IEEE Symposium on Security and Privacy*, 1990.

[NyOs 94] M. Nyanchama, S.L. Osborn. Access Rights Administration in Role-Based Security Systems. In *Database Security VIII: Status and Prospects*, 1994.

[NyOs 99] M. Nyanchama, S.L. Osborn. The role graph model and conflict of interest. In *ACM Transactions on Information and Systems Security*, 2(1):3-33, February 1999.

[NT 94] B.C. Neuman, T. Ts'o. Kerberos: An Authentication Service for Computer Networks. In *IEEE Communications Magazine*, v. 32 n. 9, pages 33-38, 1994.

[RS 95] M. Rusinkiewicz, A. Sheth. Specification and Execution of Transactional Workflows. In *Modern Database Systems*, W.Kim (ed.), 1995.

[SA 98] R. Sandhu, Gail-J. Ahn. Decentralised group hierachies in UNIX: An experiment and lessons learned. In *Proceedings of 21st NIST-NCSC National Computer Security Conference*, October 1998.

[Sal 75] J.H. Saltzer, M.D. Schroeder. The Protection of Information in Computer Systems. In *Proceedings of IEEE*, Vol.63, No 9, September 1975.

[San 90] R. Sandhu. Separation of Duties in Computerised Information Systems. In *Proceedings of IFIP WG 11.3 Workshop on Database Security*, September 1990.

[SBh 97] R. Sandhu, V.Bhamidipati. The URA97 Model for Role-Based User-Role Assignment. In *Proceedings of IFIP WG 11.3 Workshop on Database Security*, August 1997.

[SCFY 94] R. Sandhu, E.J. Coyne, H.L. Feinstein, C.E. Youman. Role Based Access Control: A Multi-Dimensional View. In *Proceedings of 10th Annual Computer Security Applications Conference*, 1994.

[SCFY 96] R. Sandhu, E.J. Coyne, H.L. Feinstein, C.E. Youman. Role Based Access Control Models. *IEEE Computer*, February 1996.

[SCFY 97] R. Sandhu, E.J. Coyne, H.L. Feinstein, C.E. Youman. The ARBAC97 Model for Role-Based Administration of Roles. In *Proceedings of 2nd ACM Workshop on Role Based Access Control*, 1997.

[SM 97] R. Sandhu, Q. Munawer. The RRA97 Model for Role-Based Administration of Role Hierarchies. In *Proceedings of 3rd ACM Workshop on Role Based Access Control*, 1998.

[SP 99] R. Sandhu, J.S. Park. Smart Certificates: Extending X.509 for Secure Attribute Services on th Web. In *Proceeddings of the 4th ACM Workshop on Role Based Access Control*, 1999.

[Ste 92] Daniel F. Sterne. A TCB Subset For Integrity and Role-Based Access Control. In *Proceedings of 15th National Computer Security Conference*, October 1992.

[SZ 97] Richard T. Simon, Mary Ellen Zurko. Separation of Duty in Role-Based Enviroments. In *Proceedings of 10th IEEE Computer Security Foundations Workshop*, June 1997.

[TA 91] J.J. Tardo, K. Alagappan. SPX - Global Authentication Using Public-Key Certificates. In *Proccedings of the 1991 IEEE Symposium on Research in Security and Privacy*, 1991.

[TJM 99] M. Thompson, W. Johnston, S. Mudumbai, G. Hoo, K. Jackson, A. Essiari. Certificate-based Access Control for Widely Distributed Resources. In *Proccedings of the 8th USENIX Security Symposium*, 1999.

[TH 90] I.A. Todd, R.D. Haggart. *The Organisation and Its Enviroment*, pages 70-90, 1990.

[Thom 91] D.J. Thomsen. Role-Based Application Design and Enforcement. In *Database Security IV: Status and Prospects*, 1991.

[TS 93] R.K. Thomas, R. Sandhu. Towards a Task-based Paradigm for Flexible and Adaptable Access Control in Distributed Applications. In *Proceedings of the Second New Security Paradigms Workshop*, 1993.

[TS 94] R.K. Thomas, R. Sandhu. Conceptual Foundations for a Model of Task-based Authorisations. In *Proccedings of IEEE Computer Security Foundations Workshop*, June 1994.

[TS 97] R.K. Thomas, R. Sandhu. Task-based Authorisation Controls: A Family of Models for Active and Enterprise-oriented Authorisation Management. In *Proceedings of IFIP WG 11.3 Workshop on Database Security*, August 1997.

[WH 97] Workflow Applications within Business Organisations, pages 41-59; Applications of Workflow, pages 89-124. In *Wokflow Handbook 1997*, P. Lawrence(ed.), John Wiley & Sons Inc., 1997.

[WL 97] Thomas Y.C. Woo, Simon S. Lam. Authentication for Distributed Systems. in *Internet Besieged: Countering Cyberspace Scofflaws*, D. Denning and P. Denning (eds.), ACM Press and Addison-Wesley, 1997.

[Zak 93] A. Zakinthinos. A Least Privilege Mechanism for User Processes. Masters Thesis, Department of Computer Science, University of Toronto, 1993.