An Objectbase Schema Evolution approach to Windows NT Security

by

Raj Kumar Jayapalan

A thesis
Submitted to the faculty of graduate studies
in partial fulfillment of the requirements
for the degree of

Masters of Science

Department of Computer Science
University of Manitoba
Winnipeg, Manitoba

THE UNIVERSITY OF MANITOBA

FACULTY OF GRADUATE STUDIES

*****

COPYRIGHT PERMISSION PAGE

An Objectbase Schema Evolution Approach to Windows NT Security

BY

Raj Kumar Jayapalan

A Thesis/Practicum submitted to the Faculty of Graduate Studies of The University of Manitoba in partial fulfillment of the requirements of the degree of Master of Science

Acknowledgements

I would like to express my sincere gratitude to my advisors Dr. Randal Peters and Dr.Ken Barker for their help and support during the course of my master's program. They provided constant support throughout this research in the form of guidance, advice and encouragement.

I should thank Dr. Randal Peters for the funding he provided throughout my masters degree.

I would also like to thank the members of the committee Dr.Randal Peters, Dr. Ken Barker and Dr. Bob Mcleod for the comments and suggestions provided.

I thank my parents, brother and sister for their affection, prayers and support without which I would not have made it this far. I also thank my friends who cheered me and gave me a ride home almost everyday late night.

I finally thank GOD for everything he has given me.

Abstract

A security model should be designed in such a way that it is transparent to the users, and at the same time easy to maintain and manage even if a very complex security model is required to ensure its proper functions. Schema evolution on the other hand is the timely change of the schema and the consistent management of these changes. Dynamic schema evolution (DSE) is the management of the schema changes while a system is in operation. The various schema evolution operations are similar to the security management operations. Thus this thesis proposes a new security model based on DSE that provides a flexible set of operations that will make security management easier and more understandable.

Windows NT is chosen as the test platform and the model is implemented on it. The current security model for the Windows NT operating system is powerful and offers many valuable features. The User Manager provided by Windows NT is the primary method for the provision of security maintenance. Our system supports the following features in addition to those currently available on Windows NT: (1) An object-oriented hierarchy, so roles and groups can be supported in a more automated way. (2) A more intuitive user interface so the administrative errors are less likely to be problematic. (3) Simplified security management on a Windows NT platform. (4) Avoids unnecessary creation of objects (users / group) and redundant granting / revoking of privileges.

One of the nicest features of the proposed security model is that both the system and the User Manager can operate together. Thus with the proposed model, the maintenance of the security model becomes much easier and more efficient.

Contents

1 Introduction **2**

1.1 Research Framework 2

1.1.1 Schema Evolution 2

1.1.2 Axiomatic Model 2

1.1.3 Security and Dynamics in Security 3

1.1.4 Role based Access Control 4

1.2 Thesis Statement 5

1.3 Assumptions 5

1.4 Key Contributions 6

1.5 Organization of the thesis 6

2 Motivating Technologies **7**

2.1 Schema Evolution 7

2.2 The Axiomatic Model 9

2.3 The Role Based Access control 10

2.4 Windows NT Security Attributes 14

3 The Axiomatic Model **19**

3.1 Overview 19

3.2 Fundamental Definitions 20

3.3 The Axioms 22

3.3 Semantics of Change 26

3.3.1 Modify Type (Add Behavior) 27

3.3.2 Modify Type – Drop Behavior 27

3.3.3 Modify Type – Add Supertype link 28
3.3.4 Modify Type – Drop a supertype link 29
3.3.5 Add a Type 30
3.3.6 Drop Type 31

4 The Windows NT Security Management System **33**
4.1 Architecture Overview 34
4.1.1 The Axiomatic Layer 35
4.2 Windows NT security Mechanisms 36
4.2.1 Add / Remove a group 36
4.2.2 Add / Remove a User 37
4.2.3 Add / Remove Member to the group 37
4.2.4 Add / Remove Privilege of a group 38
4.2.5 Add / Remove privilege from a User 38
4.3 The Security Manger Interface 38
4.4 System operation 39
4.4.1 Add / Remove group 41
4.4.2 Add / Remove a User 43
4.4.3 Modify sub-type relationship 45
4.4.4 Add/Remove Privilege of a group 46

5 Conclusion And Future Directions **49**

List of Figures

2.3.1 A sample Role Graph 13
2.4.1 The User Manager 18
3.2.1 Chain relationships 24
3.2.2 Addition of a new type 25
3.3.4.1 Effects of dropping a direct supertype link from type T to type S. 29
3.3.6.1 Effects of dropping a type T 31
4.1.1 Different Layers of Security Management System 34
4.4.1 The Interface showing groups and users 40
4.4.4.2: Interface Showing the Available Privileges 46

List Of Tables

2.3.2 Roles and their Effective privileges 13
3.1.1 Notations for axiomatic model 21
3.1.2 Axiomatization of subtyping and property inheritance 23
3.3.6.2 Axiomatic support for the proposed model 32

Chapter 1 Introduction

One of the most challenging problems in managing large networked systems is the complexity of security administration. Some of the challenges of security administration are: depth of security, ease of access, sound management, protection of integrity, cost-effectiveness, secrecy and confidentiality of key software systems, database, and data networks. Most of the security models currently used, require a trade-off between the depth of security and ease of maintenance. The dilemma is that the more secure a system becomes, the more of a barrier that security becomes to the normal operations for which it is intended. Thus a security model should be designed so it is transparent to the users. Further it must be easy to maintain and manage even if a very complex security model is required to ensure its proper functions.

This thesis proposes a new model that provides a flexible set of operations that will make security management easier and more understandable. This research uses an Object-base Management System (OBMS) because of its ability to handle the complex information with complex relationships often found in non-trivial security models. These models are often characterized by their dynamics. In other words, once a security model has been deployed, changing system and application requirements often demand that the model adapt. Fortunately, substantial research has been undertaken in recent years describing how to manage the dynamics exhibited by object-based systems. Ideally these changes to the model should occur while the system continues normal operation, as it is often undesirable or even impossible to stop the system to deploy new security policies.

1.1 Research Framework

Some of the motivating technologies utilized by the research include: Schema Evolution, Axiomatic model, Security and Dynamics in Security. This section discusses each of these motivating technologies.

1.1.1 Schema Evolution

Schema evolution has been a topic of active research for several years and many methodologies have been proposed in the past decade. Schema evolution is the timely change of the schema and the consistent management of these changes. *Dynamic schema evolution* (DSE) is the management of schema changes while a database management system or an operating system is in operation. Dynamic schema evolution, which plays a vital role in object base management systems because of its ability to make changes to the database schema while applications are running, is the major contribution of this thesis. Typical schema changes that may be required are to the domain structure, the functionality of a particular application, or to meet new performance requirements.

1.1.2 Axiomatic Model

Peters and Özsu [1] propose a sound and complete axiomatic model for dynamic schema evolution in object-based systems supporting types and inheritance. The model they propose can infer all schema relationships from two input sets that are associated with each type called the *essential supertypes* and *essential properties*. The two approaches to dynamic schema evolution are the formal and informal approaches. The axiomatic model is a formal treatment of DSE in object-base systems. Peters and Özsu [1] also describes the various dynamic schema evolution policies of TIGUKAT and indicates how these policies can be described using the axiomatic model. Özsu *et al.* [2] suggests various issues and aspects governing implementation design and development of the object model TIGUKAT.

1.1.3 Security and Dynamics in Security

Security is defined as protection against unwanted disclosure, modification, or destruction of data and applications [5]. Security not only means protecting classified information but also protecting unclassified but sensitive data or private information. The best known U.S. computer security standard is the Trusted Computer System Evaluation Criteria (TCSEC). It contains various security features and assurances that are based on the Department of Defense (DoD) security policy. The mechanism used to grant and revoke privileges is commonly referred to as "access control". The TCSEC specifies two types of access control. They are *discretionary access control* and *mandatory access control.*

In *discretionary access control*, the owner of an object has complete control over the object and specifies the access that particular users or groups can have on that object. The objects may be relations, files, named pipes, registry keys, user objects, kernel objects, and so on. There are two levels of assigning privileges. They are:

i) *Account level* where an administrator (or DBA) specifies the particular privileges that each account possesses independently on the objects.

ii) *Object level* where each object is assigned a particular privilege.

The account level privileges apply to the account in general. Privileges like create table, create schema, create file etc. are examples of account level privileges. On the other hand object level privileges apply to the individual objects. For example, specifying the access rights a user possesses on a table or file (i.e. read, write, execute) is an object level privilege.

Discretionary access control permits individual users to grant and revoke privileges at their discretion. Thus users can grant and revoke accesses to any of the objects under their control without the intercession of the system administrator. This is implemented by giving all the privileges on that object to its owner. The owner can pass privileges on any of the owned objects to other users by granting privileges to their accounts.

Mandatory access control is a means of restricting access to objects based on the sensitivity of information contained in the objects and the formal authorization of subjects to access information of such sensitivity. Mandatory access control is a multi-level access control commonly found in government, military and intelligence applications. The four main types of security classes used are top secret (TS), secret (S), confidential (C), and unclassified (U) where TS is the highest level and U is the lowest level. Such type of access control is used only for secure military systems and its use in other applications is rare [4].

1.1.4 Role based Access Control

Ferrialo and Cugini[5] present a non-discretionary access control known as role based access control (RBAC) that is more central to the secure processing needs of non-military systems. According to Ferrialo and Kuhn[4], access control decisions are often based on the roles individual users take on as part of an organization. A role is said to specify a set of transactions that a user or set of users can perform within the context of an organization. The key components of the RBAC model are users, roles, operations and sessions [7].

Multi-user operating systems like UNIX and Windows NT have a poor interface for security management. Hua and Osborn [9] modeled UNIX access control with a role based access control. The role graph is one manifestation of RBAC. A role graph is used to visualize the permissions granted to the files in the UNIX system. However, to completely model the existing permissions in a UNIX environment, the system file permission and the links between the files must be modeled too.

Although role based access control has been realized with UNIX and a few other environments[14], it cannot be implemented in the Windows NT system as Windows NT does not allow a complete non-DAC oriented access control mechanism

Thus the role-based mechanisms cannot be implemented on the Windows NT operating system. To provide RBAC a new security model must be implemented that provides a better visual interface and easier security management while continuing to provide the existing features of the system.

An object-oriented axiomatic model that captures schematic evolution is adapted to improve system security dynamically. Dynamic Schema Evolution provides an effective solution to this problem in object-based systems so the same feature is used in the proposed security model.

1.2 Thesis Statement

This thesis proposes a security management model based on well known schema evolution techniques in OBMSs. The Windows NT operating system is taken as a case study and the proposed security model is implemented on it. The proposed model is implemented in the Windows NT operating system in such a way that no changes are done to the underlying operating system. Further, these changes can occur without the need to reboot the system. The model permits the user to perform new operations that makes the management of security easier and more understandable.

1.3 Assumptions

The following assumptions are made in this thesis:

- In Windows NT the users cannot grant privileges to other users because only the system administrator can grant/revoke privileges. In short Windows NT security management requires "administrator" privilege to change access rights even though the object may belong to another user.
- The proposed model will be implemented only on non-military systems with single level access control.
- Windows NT does not support mandatory access control so the issue is beyond the scope of the thesis.
- Windows NT supports discretionary access control and is considered.

1.4 Key Contributions

This thesis contributes by

- Providing an object-based security model based on the axiomatic model's access control mechanism.
- Providing a security model that can be implemented without changing the operating system.
- Providing a security model that prevents redundant granting/revoking of privileges. It also avoids unnecessary creation of groups thereby simplifying security management.
- Providing an intuitive interface for the security model that simplifies the task of security administrator.
- Granting no more privilege than necessary to perform a task. This property ensures the adherence to the principle of least privilege.

The model will be compared with the current Windows NT security model to make sure that it supports all the functionality that the current model supports. It is also compared with the axiomatic model to ensure that its approach is axiomatic.

1.5 Organization of the thesis

The remainder of the thesis is organized as follows: In Chapter 2 a review of related work in schema evolution, security and role-based access control is presented. In Chapter 3, an overview of the axiomatic model of dynamic schema evolution is presented. The similarities and difference between the schema evolution and security management is also discussed. Chapter 4 describes the architecture of the proposed security model based on schema evolution. The advantages of this model over the native security model of Windows NT are also discussed. Chapter 5 contains the conclusion and future work that should be considered.

Chapter 2 Motivating Technologies

By applying Schema Evolution to the Windows NT security model the following benefits should be realized:

1) It greatly simplifies the management of user/group privileges.
2) It also provides greater flexibility and
3) Object-oriented approach for security maintenance.

Schema evolution has been investigated intensively recently [1,2,3] but its potential application to security management has not been investigated to date.

Three fundamental aspects need to be considered before we can address the results achieved to date. The first of these is the work on schema evolution in object-oriented systems with particular focus on an axiomatic model. Secondly, work directly related to non-discretionary access rules that are often captured in roles. Finally, it is useful to consider other work attempting to provide a "new" interface to an existing systems security model. Each of these is briefly discussed below.

2.1 Schema Evolution

Schema evolution captures the changes to the schema and ensures that these changes are managed consistently. Dynamic schema evolution (DSE) is the management of schema changes while a system is in operation. Schema evolution is very important to object-oriented databases because post design modification is a common phenomenon. The various post design changes[1] addressed to date include: changes in domain structure, changes in the functionality of a particular application, and changes needed to meet the performance requirements.

The object-base management system's client application may be changed depending on how the information is organized. All such changes need to be tracked by these applications. Some of the common schema update operations include:

- Create a new object type
- Delete an existing object type
- Add a subtype link between existing object types
- Delete a subtype link between existing object types
- Modify type: Add a new property
- Modify type: Delete a property
- Modify type: Change an existing property

Peters and Özsu [1] argue about two fundamental problems in schema evolution:

1. *Semantics of Change:* The effects of the change on the overall way in which the system organizes information (i.e. the effects on the schema).
2. *Change propagation:* The effects of the change on the consistency of the underlying objects (i.e. the propagation of the changes to the existing instances).

Screening: The problem of semantics of change is discussed with screening. In this method a conversion program is generated in response to schema changes and these programs are capable of converting the objects into new representation independently. The second solution to the same problem is *conversion* where each schema change initiates an immediate conversion of all the objects affected by the change. Both these solutions cause a delay either during accessing the objects (screening) or during the modification of schema (conversion).

Filtering is a technique that can solve the problem of change propagation. According to this technique all the affected objects are updated thereby changing their representation as dictated by the new schema.

A model can use either *screening*, *filtering*, or both. The problem of Semantics of Change and Change Propagation need to be addressed when schema evolution is applied to the security model. The system should update each of the object's (i.e. users/groups) attributes in response to any changes in the properties or its relationship with other objects.

Several researchers have investigated schema evolution in object oriented databases[1,2,3]. Orion[15], Gemstone[16] and O_2[17] are some of the commercially available object-oriented database management systems that support schema evolution. Evolution is defined in terms of operations that change individual type definitions. O_2 provides high level operations to manipulate class hierarchies and provide better support in expressing type changes while preserving data integrity.

A formal treatment of schema evolution in object oriented databases is presented by Peters and Özsu [1]. An axiomatic model is introduced that provides a solution for dynamic schema evolution by serving as a common, formal underlying foundation for describing schema operations. This common framework makes the task of schema comparisons easier. The authors illustrate the dynamic schema evolution in Tigukat and Orion using their axioms.

2.2 The Axiomatic Model

Peters and Özsu [1] proposed the axiomatic model that provides a solution for dynamic schema evolution. Schema evolution could be addressed either informally or rigorously using a formal methodology. Informal approaches lead to multiple dynamic schema evolution mechanisms because of the differences in the object model and the choices made by the system designers [1]. This results in the lack of a common object model that makes the comparison of the object-bases difficult.

Peters and Özsu's axiomatic model [1] is a formal treatment of dynamic schema evolution in object-base systems. This model provides a solution for dynamic schema evolution in an object-based system by serving as a common, formal, underlying foundation for describing dynamic schema evolution of existing systems. The major benefits of the axiomatic model for dynamic schema evolution in the object-based systems are:

- The model is a formal specification of dynamic schema evolution in object-base systems.
- The model has proven soundness, completeness and termination properties for the derivations performed.

- The model is powerful enough to express the dynamic schema evolution of existing objects.
- The model can be used in practice by serving as a common foundation for characterizing and comparing the dynamic schema evolution of various systems.

This thesis contributes by describing how the axiomatic model can be applied to a security model. Schema evolution has many similarities to security maintenance. Security management undergoes dynamic changes and all these changes need to be updated, while the security system is in operation. The Windows NT or UNIX systems for example have the concepts of groups and users. Since the axiomatic model treats all the objects as types, the concept of users or groups can be completely replaced by the term type. The axiomatic model is object-oriented and complete so an efficient security model can be implemented with it.

Peters and Özsu [1] formalized the dynamic schema evolution characteristics into a well defined set of axioms. The axioms automatically maintain the complex schema relationships and properties of the schema. Their model can infer all schema relationships from two input sets *essential supertype* and *essential properties*. The user, schema designer, system or a combination of sources can provide the elements of these sets. These axioms were consistent with the security management of many systems so they are implementable. More details about the axiomatic model will be discussed in the next chapter.

2.3 The Role Based Access control

RBAC has existed for over twenty years, but renewed interest has occurred recently because of the disenchantment with the traditional mandatory and discretionary access control by many users. The major goal of the role based access control is that, users do not have discretionary access to enterprise objects. Ferrialo and Kuhn [4] argue that non-discretionary access control (role based access control) is critical to the secure processing needs of non-military systems. The major advantage of this mechanism is that the users are made members of roles (depending on their qualifications and responsibilities) and

can be easily reassigned from one role to another without modifying the underlying access structure.

Barkley [7] compared the role based access control mechanism (RBAC) with access control lists (ACL). The various advantages of the role based access control mechanism over access control list are also discussed. One of the most important features of role based access control is that it is not necessary to translate a corporate organizational view into another view to accommodate an access control mechanism. It should be noted that a simple RBAC model is similar to ACL in its expressive power. The only difference is that RBAC allows a session (a set of processes called subjects that act on behalf of the user) to be associated with a proper subset of the roles (groups in ACL) authorized for a user.

Hua and Osborn [9] provides an interface between role based access control and UNIX. A model of how to access UNIX files using the role based access control is also described. A role graph is used to visualize the permissions granted to the files in the UNIX system as shown in Figure. 2.3.1. However, to completely model the existing permissions in a UNIX environment, the system file permission, and the links between the files must yet be modeled.

According to Hua and Osborn [9] the role graph model is based on a very general notion of privileges. A privilege is a pair (x,m) where x refers to an object and m is a non-empty set of access modes for an object x. The object referenced by x can be a database element in a database environment or any system resource whose access need to be controlled.

The access modes m can be any valid operations on x. For example, in Unix m is an element of the access modes read, write or execute. The role can be defined as a named set of privileges. Hua and Osborn [9] represent role as (*rpname*, *rpset*) where *rpname* is the name of the role and *rpset* is the set of privileges that the role possess. Given a role r, *r.name* refers to the name of the role while *r.rpset* refers to the set of privileges of the role. The terms supertype and subtype used in the axiomatic model use the analguous terms *is-senior* and *is-junior* respectively, thus the role r_i is *is-junior* to r_j, if r_i. *rpset* $\subset r_j$. *rpset*. Conversely r_j is *is-senior* to r_i.

There are two kinds of privileges found in a role graph. They are:

Direct Privilege: These are privileges given to the role directly. A direct privilege of a role r are those which are not contained in the *rpset* of any of r's immediate juniors.

Effective Privilege: These are the union of both the direct and effective privileges of all its juniors.

Every role graph has at least two roles:

MaxRole – A union of all the privileges of the roles in the role graph i.e. the role with the maximum privilege

MinRole – the minimum set of privileges available to all the roles. It is possible for the MinRole to have no privileges.

Some of the important properties of a role graph include:

- A single MaxRole since MaxRole is the root of the role graph that summarizes all the privileges in the system.
- A single MinRole since MinRole is the base of the system that possesses the minimum privilege and all the roles inherit those privileges.
- The role graph is acyclic thereby disallowing any role r_i to be both *is-junior* and *is-senior* of a role r_j.
- There is a path from r_i and r_j iff $r_i.\ rpset \subset r_j.\ rpset$

Consider the role graph in Figure 2.3.1 [9].

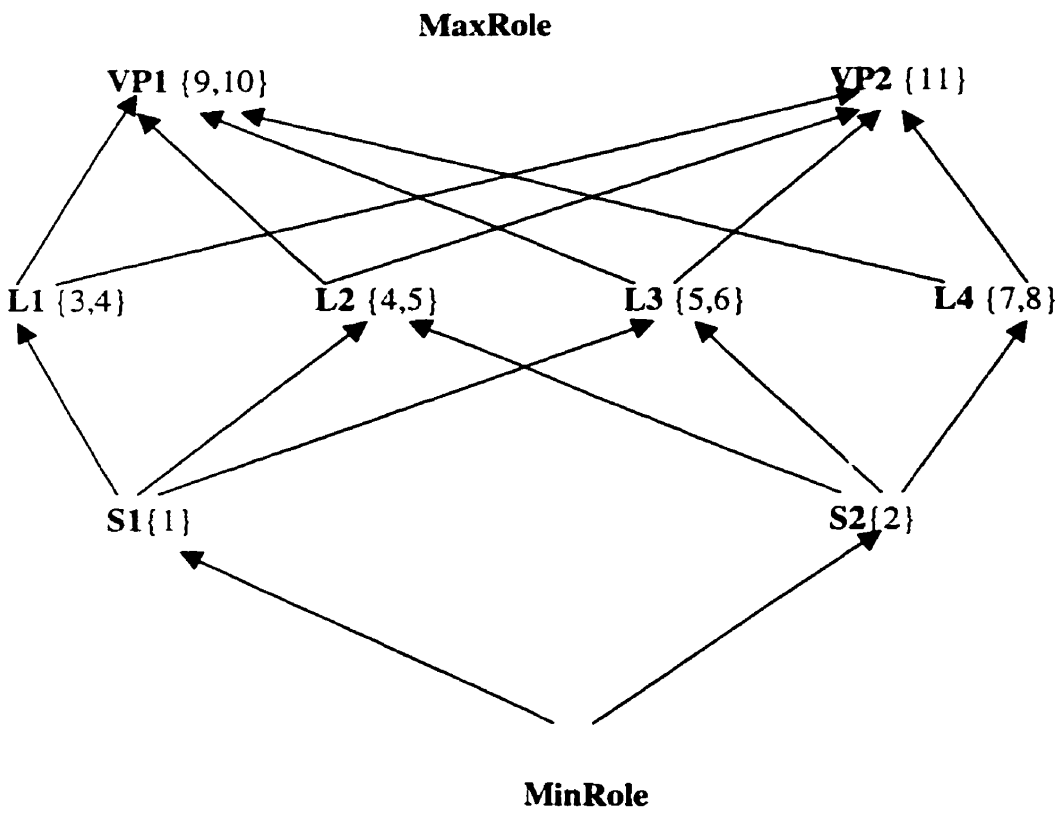


Figure 2.3.1 A sample Role Graph

Table 2.3.2 shows both the direct and effective privileges for each role.

Role Name	Direct Privileges	Effective Privileges
MaxRole	Φ	{1,2,3,4,5,6,7,8,9,10,11}
VP1	{9,10}	{1,2,3,4,5,6,7,8,9,10}
VP2	{11}	{1,2,3,4,5,6,7,8,11}
L1	{3,4}	{1,3,4}
L2	{4,5}	{1,2,4,5}
L3	{5,6}	{1,2,5,6}
L4	{7,8}	{2,7,8}
S1	{1}	{1}
S2	{2}	{2}
MinRole	Φ	Φ

Table 2.3.2 Roles and their Effective privileges

Role graphs have successfully modeled access control mechanisms in UNIX [9]. Role graphs cannot be used to model the Windows NT security mechanisms because it does not directly support role hierarchies, does not directly support role constraints nor does it allow a complete non DAC oriented access control mechanism. Moreover the users cannot have different sets of permissions for different roles. Smith *et al.* [14] carried out various analysis to determine if RBAC the mechanism could be implemented in Windows NT 3.5, Oracle 4 and Novell Netware 4.1. They concluded that role hierarchies could not be implemented in Windows NT 3.5 and Novell Netware 4.1 but could be implemented successfully in Oracle 7. Since Windows NT does not support a completely non-discretionary access control mechanism, this thesis concerns with the Smith *et.al.*[14] findings.

2.4 Windows NT Security Attributes

The three most important components of Windows NT are *Users*, *Groups* and *Domains*. This section briefly reviews the important aspects of these Windows NT objects to get a better understanding of these objects because these terms are extensively used in the remaining chapters of the thesis. More extensive Windows NT documentation have been produced including the contribution of Minasi *et al.* [10].

Users : For our purposes a user is a person who uses the system. Each user has an account and a small database record of information about the user is stored in the system. Every user should have a user name, user password and user restriction or user permission and rights.

Domain: A domain is a collection of computers. NT allows a group of machines (connected on the network) to centralize and share the database (that contains information about the users) on a single machine. Thus whenever a user needs to log on from any one of the machines, NT searches for the account information of the user on the database on the central machine. The machine that holds the central database is called the Primary Domain Controller (PDC). The PDC maintains a database of information about all the users on the domain. The information retained includes the user name, their passwords, the groups to which each user belongs and the privileges or rights they possess.

Groups : A group is a collection of users. The main objective of groups is to simplify the process of administration as it is easier to specify security rules to a single group rather than dozens of individual members within the group. There are two different kinds of groups:

- Global Groups
- Local Groups

Global Groups: A global group is a collection of user accounts that are visible to any account participating in a domain. They contain user accounts from one domain. The most important concept is that the global group cannot be a container for another global or local group. Thus a global group is just a collection of individual user accounts.

There are three predefined global groups that Windows NT creates by default. They are:

- *Domain Admins*: Members of Domain Admins can administer the home domain, the workstations of the domain, and any other domains that have added the Domain Admins global group to their own administrators local group.
- *Domain Users*: Members of this group have normal user access to both the domain itself and for any NT workstation in the domain.
- *Domain Guests*: This group allows guest accounts to access resources across domain boundaries, if they are granted the required privileges by the domain administrators.

All these global groups do not exist on the local systems. They only apply to Windows NT domains set up on a Primary Domain Controller.

Local Group: A local group applies only to a local system and is basically a named collection of user accounts (both global and local) and global groups. They contain user accounts created on the local computer as well as user accounts and global groups from the domain to which the local computer belongs. Local groups can contain global groups but local groups cannot contain other local groups.

There are six predefined local groups:

- *Administrators*: The members of this group can fully administer the computer.
- *Backup Operators*: The members of this group can bypass the file security to back up files.
- *Guests*: This group is set up mainly for users who log on occasionally. Thus very limited access is given to members of these groups.
- *Power Users*: The users under this group can share directories and printers.
- *Replicator*: The members of this group can support file replication in a domain.
- *Users:* The members of this group are ordinary users who have minimal rights.

We now turn our attention to the different kinds of privileges and their properties that exist in the Windows NT System.

Privileges or User rights: A privilege or user right is the ability to do a task such as backing up data on a server or logging on to the local machine, *etc.* It is these privileges that distinguish the different groups (or users). Windows NT has a pre-defined non-extensible set of 27 privileges. Often used rights in Windows NT are:

- Access this computer on the network
- Add workstations to the domain
- Back up files and directories
- Change the system time
- Force shutdown from a remote system
- Load and unload device drivers
- Log on locally
- Manage auditing and security log
- Restore files and directories
- Shut down the system
- Taking ownership of files or other object

In addition to these there are other advanced user rights. Each of these user rights define a specific activity that may be performed once the privilege is enabled. Each of these privileges has three attributes: a programmatic name, a display name, and a local unique identifier. For example the privilege "Force Shutdown the System" is the display name and the corresponding programmatic name is 'SeRemoteShutdownPrivilege'.
We now focus on the tool that is used to maintain the security in NT. The various operations that can be done and some of the limitations of the tool will also be discussed.

The User Manger :

The user manager and the user manager for domains are the tools provided by Windows NT to administer the user accounts. The user manager for domains administers the domains while the user manager administers the local computers.
Some of the operations that could be done with the user manager include:

- adding a new group
- adding a new user
- deleting a group
- deleting a user
- adding a member to a group and
- deleting members from the group

The user manager lacks some key features such as providing a list of users who possess a particular privilege, list privileges that were inherited from their parents, and an intuitive interface. Other user manager shortcomings will be discussed in Chapter 5.

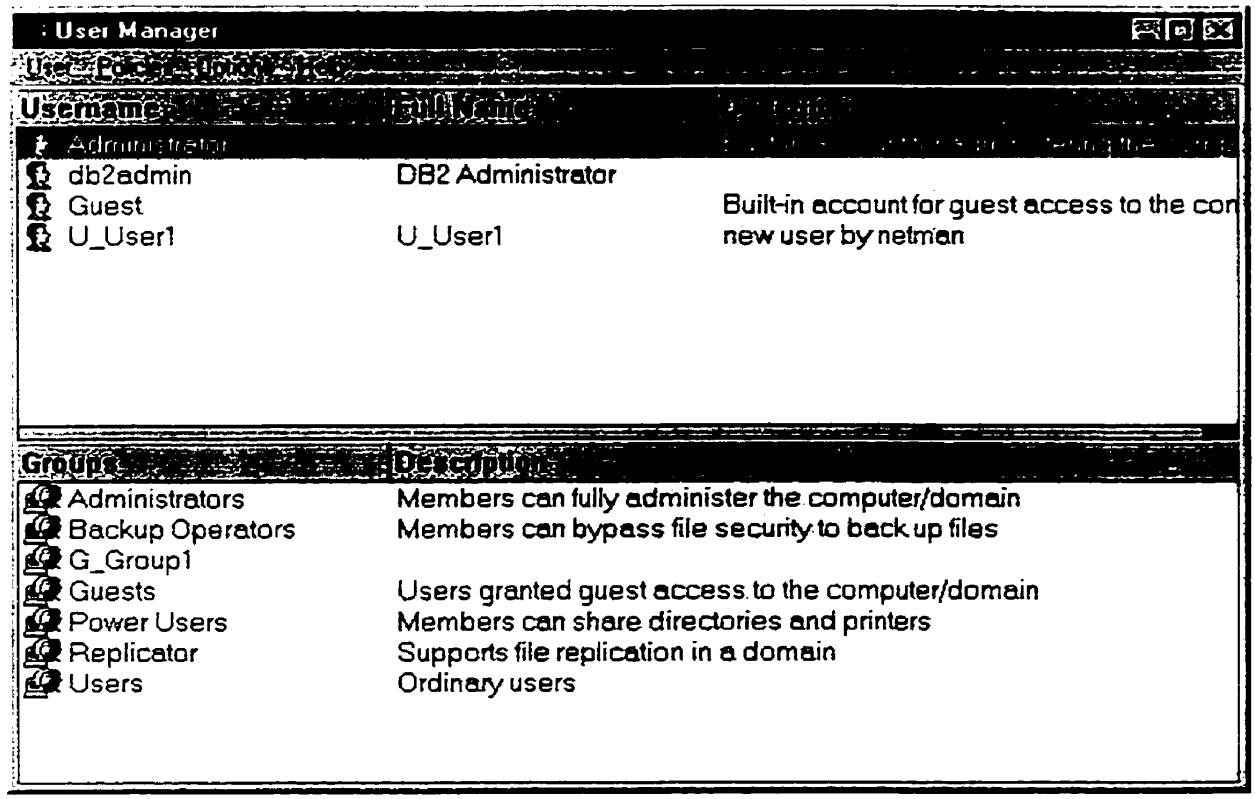


Figure 2.4.1 The User Manager

Figure 2.4.1 illustrates the User Manager. *Administrator* and *Guest* are system defined users while *db2admin* and *U_User1* are created by the User Manager. Similarly the system defined groups include *Administrators*, *Backup Operators*, *Guests*, *Power Users*, *Replicator* and *Users* and the group *G_group1* is newly created.

Chapter 3 The Axiomatic Model

3.1 Overview

This chapter discusses how the schema evolution operations relate to the axiomatic model, the similarities between the security management and schema evolutions and how the axiomatic model could be used to manage the security. For better understanding the various definitions and axioms associated with the axiomatic model are defined. The remainder of the chapter presents the schema evolution operations and their relation to the security management.

A security model that treats all its entities (like the users or groups) uniformly will ease maintenance. The axiomatic model is object-oriented and solves the problem of schema evolution. The proposed approach uses the axiomatic model to maintain the security of various systems. The approach's utility is demonstrated with Windows NT.

The axiomatic model supports features such as subtyping and inheritance. The model is a formal specification of dynamic schema evolution in object-base systems. All the characteristics of schema evolution are formalized into a well defined set of axioms that automatically maintains the complex schema relationships and properties.

When this model is used for security, a mapping of all the security components (like users, groups, privileges, *etc.*) with the components of the axiomatic models is required. Any changes in the security components are subsequently translated to the corresponding schema changes in the axiomatic model. The axiomatic model can automatically update the complex relationships and properties so the resulting changes are propagated to the security components of the system as in Figure 4.1.1. The mapping of the security components to the axiomatic components depends on the type of the system on which the model is implemented. Thus by changing these mappings the same

model can be implemented for different systems. Before arguing about correctness, we must define some key terms in the axiomatic model.

3.2 Fundamental Definitions

There are two input sets associated with each type in a schema used by the axiomatic model, namely: essential supertypes and essential properties.

Essential supertypes are denoted by $P_e(t)$ where t is a type in a schema. The essential supertype of a type contains all the types that must be maintained as supertypes of *t* as long as it is consistently possible. The only way to break a link from *t* to an essential supertype *s* is to explicitly remove *s* from $P_e(t)$ by either dropping the subtype relationship between *t* and *s* or by dropping *s* entirely.

Essential Properties are denoted by $N_e(t)$. The essential properties are those properties identified as being essential to the construction and existence of type *t*. The essential properties of a type consist of all properties natively defined by the type and may contain properties inherited from its supertypes.

Several Additional sets are defined and maintained as part of the axiomatic model because of their usefulness. It should be noted that these additional sets are entirely derived from the essential supertype and properties sets.

Native Properties: The native properties N(t) of a type *t* are properties that are not defined in any of the supertypes of *t*.

The Inherited properties H(t): of a type *t* is the union of the properties defined by all supertypes of *t*.

The Interface I(t): is the union of the native properties N(t) and inherited properties H(t). The interface set serves as a specification of all properties maintained by a type.

Subtyping ( ): allows types to be built incrementally from other types. The common notation used for subtyping is . The notation *t* *s* denotes that *t* is a

subtype of *s*. A subtype inherits all the properties of its supertype and can define additional properties that do not exist in the supertype.

Type Lattice L = (T, ≤) consists of a set of types T together with a partial order ≤ of the elements of T based on the subtype relationships ( ).

The Supertype Lattice types (PL(t), $\leq_t$) is the set that includes *t* and all supertypes (including both immediate and essential) of *t*. The type lattice L_t of a type *t* is the set PL(t) including *t*, with a partial order $\leq_t$ of all the types based on the subtype relationships. This is known as *supertype lattice* and is a subset of the type lattice L.

The *apply-all* operation $\alpha_x(f,T')$ is used to support the axioms. This operations applies the unary function *f* to the elements of a set of types $T' \subseteq T$ and the function f is defined over a single variable *x*. The semantics of the apply-all operation is to let *x* range over the elements of T' and for each binding of *x*, evaluate *f* and include the result in the final result set.

Term	Description
T	The set of all types of a system
L	The type lattice of a system
$s,t,t_I,T,\perp$	Type elements of T
P(t)	Immediate supertypes of type *t*
$P_e(t)$	Essential supertypes of type *t*
PL(t)	All supertypes of type *t*
L_t	Supertype Lattice of type *t*
N(t)	Native properties of type *t*
H(t)	Inherited properties of type *t*
$N_e(t)$	Essential properties of type *t*
I(t)	Interface of type *t*
$\alpha_x(f,T)$	Apply all operation

Table 3.1.1 Notations for axiomatic model

The required notation is summarized in the Table 3.1.1.

3.3 The Axioms

The axioms that are used in the axiomatic model are as follows:

Axiom of Closure: All types in T have supertypes in T giving closure to T.

Axiom of Acyclicity: There are no cycles in the type lattice formed from T and its partial order.

Axiom of rootedness: A single type $\top$ in T exists and is the supertype of all types in T.

Axiom of pointedness: A single type $\perp$ in t exists and is the subtype of all the types in T.

Axiom of supertypes: The set of immediate supertypes of a type t is exactly the subset of the essential supertypes that cannot be reached transitively through some other type.

Axiom of Supertype Lattice: The supertype lattice of a type t includes itself and recursively all supertypes of t formed from the supertype lattices of its immediate supertypes.

Axiom of interface: The interface of a type consists of the union of the native and inherited properties of that type.

Axiom of Nativeness: The native properties of a type are the subset of the essential properties that are not inherited.

Axiom of Inheritance: The inherited properties of a type t is the union of the interfaces of its immediate supertypes.

The Table 3.1.2 lists the set of axioms used to guide schema evolution.

Axiom of Closure	$\forall t \in T,\ P_e(t) \subseteq T$
Axiom of Acyclicity	$\forall t \in T, t \notin \cup\ \alpha_x(PL(x), P(t))$
Axiom of Rootedness	$\exists\ T \in T, \forall t \in T \mid T \in PL(t) \wedge P_e(t) = \{\ \}$
Axiom of Pointedness	$\exists\ T \in T, \forall t \in T \mid t \in PL(\perp)$
Axiom of Supertypes	$\forall t \in T, P(t) = P_e(t) - \cup\ \alpha_x(PL(x) \cap Pe(t) - \{x\}, Pe(t))$
Axiom of Supertype Lattice	$\forall t \in T, PL(t) = \cup\ \alpha_x(PL(x), P(t)) \cup \{t\}$
Axiom of Interface	$\forall t \in T, I(t) = N(t) \cup H(t)$
Axiom of Nativeness	$\forall t \in T, N(t) = N_e(t) - H(t)$
Axiom of Inheritance	$\forall t \in T, H(t) = \cup\ \alpha_x(I(x), P(t))$

Table : 3.1.2 Axiomatization of subtyping and property inheritance

These axioms can be used to present a uniform framework for specifying the semantics of the schema and how it evolves. Furthermore these axioms also serves as a foundation for developing the security model.

Typical schema changes like adding and dropping types, adding and dropping sub/supertypes relationships between types, and adding and dropping properties of a type have similarities to the operations performed by the security model. These operations include adding/deleting user, adding/deleting membership relationships between users and groups, and adding/deleting user rights or permissions. The axiomatic model provides a good solution to schema evolution and thus this chapter discusses how this axiomatic model can improve the security of different systems.

3.2 Invariants of schema

Peters and Özsu identified a set of invariants for maintaining the semantics of schema modifications in TIGUKAT [1]. These invariants must hold before and after schema changes are performed. The invariants include type lattice, full inheritance, domain compatibility, distinct behavior, full implementation and direct supertype invariants. These invariants are applicable to security maintenance too, so each is briefly discussed below.

Type Lattice Invariant: The type lattice is a connected directed acyclic graph. Thus some of the important terms of graph theory are defined in terms of the Type Lattice. The *nodes* of the lattices are types and the *edges* are subtype relationships. In the acyclic graph shown in Figure 3.2.1, the tail of the edge is the subtype of the type pointed to by the head. In the proposed security model, the direction of the arrow is changed for the sake of clarity so the tail is the supertype of the type pointed to by the head. The lattice has a single system defined type T_object as its root and the system defined T_null as its base. There are no isolated types and thus every object in the lattice is a subtype of T_object.

Chain: A chain in the type lattice is a collection of types, totally ordered by subtyping, such that they form a single connected path through the lattice. A chain can also be defined as a collection of types that are connected by sub/supertype relationships such that they form a connected path through the lattice.

A chain of length one from a type T_a to a supertype T_b is called a direct supertype link from T_a to T_b or a direct sub_type link from T_b to T_a as shown in Figure 3.2.1.

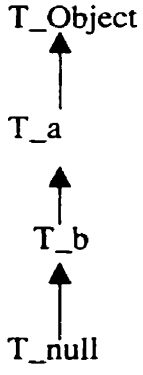


Figure 3.2.1 Chain relationships

Full inheritance Invariant: A type inherits all the behaviors from its supertypes. All the behaviors inherited by a type are called the inherited behaviors of the type. The additional

behaviors that are defined by the type are called the native behaviors. The union of the native and inherited behaviors is called the interface of the type. Thus, a type's interface is a superset of the union of interfaces of its supertypes.

Direct Supertype Invariant: A direct supertype link between two given types T_a and T_b is the only chain linking the types. This is because, if another chain links both the types, then this direct supertype link is dropped as shown in Figure 3.2.2. This means that, if there exists a chain from T_a to T_b greater than length one, then there are no direct supertype links (i.e. chains of length one) from T_a to T_b.

Figure 3.2.2 Addition of a new type

Domain Compatibility Invariant: The result type of a behavior in a type must generalize the result type of that behavior in all subtypes. That is, the result type of a behavior defined on a type, say T_a must generalize the result type of that behavior in all subtypes of T_a.

Temporal Invariant: The behaviors defined in the interface of a type at a given time are applicable to all instances of that type that exist at that time. Thus, if a behavior exists in the interface of a type at a given time t and t is within the life span of an object of that type, then the behavior is applicable to the object.

Recall that the two issues of schema evolution are the semantics of change and change propagation. Semantics of change refers to the effects of the schema change on the overall way in which the system organizes information (i.e. the effects on the schema). Conversely change propagation refers to the method of propagating the schema change to the underlying objects (ie. to the existing instances). The next section describes how the axiomatic model deals with these two problems.

3.3 Semantics of Change

This section shows how the Axiomatic Model solves the problem of Semantics of Change and Change Propagation. It also shows how this model links to the security model and the impact that it has in doing so.The typical schema changes includes:

- Modify Type (Add Behavior)
- Modify Type (Drop Behavior)
- Modify Type (Drop supertype Link)
- Add Type
- Drop Type
- Add Class
- Drop Class
- Add Collection
- Drop Collection

Among these schema changes add class, drop class, add collection and drop collection are not discussed as all the security objects like the users or groups are viewed as types and thus these schema changes are not used. This section considers all other type related operations.

3.3.1 Modify Type (Add Behavior)

This operation adds a behavior to the type. The operation cannot be carried out if it is already defined natively. This is mainly because the behaviors must be distinct in the axiomatic model. The behavior is inherited by all subtypes of the type to which it is added. However if this behavior is already inherited by any of its subtypes then there is no change in its behavior. The event corresponding to this schema change (adding a behavior) in the security model is the granting of privileges. Any user or group can be granted privilege only once. Thus granting of privileges to a group or user should be rejected if the user or group has already obtained the privilege natively. The proposed model will allow the addition of a privilege to a user or group, if the privilege is not native. However the privilege is only added to the set of native privileges and is not granted if the privilege is already inherited from some other objects (user or groups). Similarly if the privilege is already acquired by any of its subtype through inheritance then it results in no change in privilege.

3.3.2 Modify Type – Drop Behavior

This operation drops a native behavior from a type. The operation cannot be carried out if the behavior is not defined on the type or if it is inherited from another type. Thus, only native behaviors can be dropped. If an inherited behavior of a type must be dropped then the corresponding behavior should be dropped from all its supertypes. Otherwise, the full inheritance invariant will cause the type to re-inherit the behavior once again. Thus with the limitation that only native behaviors can be deleted, the original behaviors of all the supertypes remains unchanged.

When a native behavior is dropped, its native definition is propagated to all its subtypes. If the subtype inherits this behavior through some other chain, then the behavior is added to the set of inherited behaviors otherwise this behavior becomes native to that type. With this approach, the interface of the subtypes, retain all their original behaviors and only the single affected type (whose behavior is removed) drops that behavior. This approach is taken in the TIGUKAT object model because it allows

behaviors in a type to be flagged as semi-native. Thus, such behaviors should not be dropped by a recursive decent drop process but instead should persist, as native definitions in those types. In the case of the ORION object model, the semantics of behavior dropping is to recursively drop the behavior from all the subtypes as well. This is similar to the operation of dropping of privileges in the management of security. The approach of ORION is taken when this model is implemented for the maintenance of security as the TIGUKAT approach will make the security management more complex. In security maintenance, when a privilege is removed from the group, then this privilege should be removed from all its subgroup (users). Thus the approach taken by the ORION object model is used to ease security management.

3.3.3 Modify Type – Add Supertype link

This operation adds a subtyping relationship between two types. A type A cannot be added as a supertype of type B if

- It introduces a cycle in the lattice
- A is already linked to B through some other chain
- There exists a behavior *b* found in both A and B and the result type of the behavior in A does not generalize the result type in B.

The third rule is not taken into consideration when implementing the security model as the behaviors are treated as privileges that does not have a result type.

When A is added as a direct supertype B, all the behaviors of A are inherited by B and all its subtypes. This is equivalent to propagating the inheritance of added behaviors defined above and follow all the rules established for that operation.

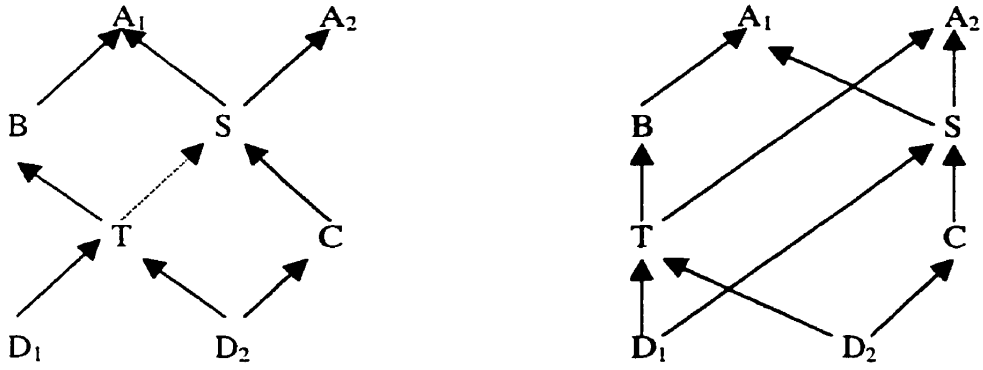
The privilege propagation is a feature that would simplify the process of security maintenance. Thus when any object is added as a subtype of an object, it inherits all the privileges thereby avoiding redundant granting of privileges. This schema change is similar to adding a user as a member of a group.

3.3.4 Modify Type – Drop a supertype link

This operation drops a subtype link between two types. A direct supertype link to T_Object cannot be dropped. The semantics of the operation are:

- to drop the direct supertype link between the type and the direct supertype of that type
- re-establish links between the type and supertypes of the direct supertype
- re-establish links between the type and the subtypes of the direct supertype

Consider the example illustrated in Figure 3.3.4.1.



Before Dropping link between T and S

After Dropping the link between T and S

Figure 3.3.4.1 Effects of dropping a direct supertype link from type T to type S.

Now assume that the direct supertype link from T to S is to be removed. The following set of operations is carried out in the process of deleting the subtype link:

- According to the axiomatic model, a supertype link from T to every supertype of S is added unless T is linked to the supertype(s) through another chain. Thus in the figure, T is re-linked to A_2 but not to A_1 since T is already linked to A_1 through the chain containing B. Unfortunately this approach cannot be implemented in the security model. Thus this semantic change will not add the link between T and A_2 when implemented for security. In this case A_2 not only

loses the direct supertype but also all the supertypes of the direct supertype T unless they are linked to this type through some other type or T is the essential supertype of A_2.

- This schema change will add a supertype link from each subtype of T to S, unless the subtype is linked to S through another chain. Thus D_1 is re-linked to S, but D_2 is not since it is already linked to S through the chain containing C. This ensures that the interface of T's subtypes are not affected by the change. In the proposed model implementing this feature makes the process of security management more complex and thus D_1 is not re-linked to S. Thus S loses all the subtypes of T unless the subtypes has S as their supertype.
- This operation drops the native behaviors of S from the interface of T. In the axiomatic model, the behaviors are not dropped from the subtypes of T, as the subtypes are re-linked to S by the step above and therefore inherit its behaviors. Since the above two steps are not implemented in the proposed security model all the behaviors (both inherited and native) are removed from the type, unless the behaviors are native or inherited through some other chain.

The above operation must be slightly modified to suit the needs of the security environment. Dropping a supertype link can be compared to the removal of membership of a user from a group. When a user loses the membership in the group, then the user should also lose all the privileges that comes from the group. Windows NT handles the privilege propagation in this way and thus the proposed model is also implemented with the same features. This change however does not affect the correctness of the axiomatic model.

3.3.5 Add a Type

This operation creates a new type and adds it to the existing lattice. Type creation is also supported through regular subtyping which is an operation provided by the axiomatic model. The axiomatic model allows the creation of a type with specific behaviors.

The proposed security model views all security attributes, like the users and groups, as types. Thus creation of a new user or group is similar to the schema change of addition of a type. The proposed security model views both the groups and users as one

single entity type, thus a user can be a supertype of both users and groups and similarly a group can be a supertype of both users and groups. Certain operating systems like the Windows NT do not allow the addition of user to be a supertype of any object. This results in the redundant creation of new groups, addition of privileges, and addition of the members to that group. These operations add more complexity but when all the entities (users and groups) are treated uniformly the management of security becomes more efficient.

3.3.6 Drop Type

This operation drops a type, thereby removing it from the schema. It should be noted that the primitive types of the model cannot be deleted. Let us consider the example shown in Figure 3.3.6.1.

Before Dropping T

After Dropping T

Figure: 3.3.6.1 Effects of dropping a type T

According to the axiomatic model, every direct subtype (B_j) of a dropped type T is re-linked to every direct supertype (A_i) of T unless there is a chain from B_j to A_i that does not include T. Furthermore the native behaviors of T are propagated to the direct subtypes so that they become native in the subtypes unless the behavior is inherited through some other chain. Recall however that this could increase the complexity of security management. Thus the same model is modified when used for security maintenance. In the proposed model every direct subtype of B_j of a dropped type T is removed from every direct supertype A_i of T unless there is a chain from B_j to A_i that

does not include T. The native behaviors of T are removed from every subtype of T unless the behavior is inherited through some other link. This object group is formed because any change to it should be propagated to all its members thereby avoiding redundant granting and revoking of privileges to each user.

Thus according to the axiomatic model the subtype B_1 is re-linked to both the supertypes A_1 and A_2, while B_2 is re-linked to A_1 but not A_2 as it is already linked to A_2, through the chain that includes S. Thus in the proposed security model such re-linking is not possible.

In the axiomatic model if both T and S define a native behavior *b*, then a native definition of *b* is propagated to B_1 and not B_2 as the behavior is still inherited through S. However in the proposed model the privilege is removed from B_1, as the object obtained this privilege from T and since T is removed the privilege does not exist any more. Thus the privilege *b* continues to be an inherited privilege in the case of B_2.

In short with certain modification to the axiomatic model, it can be successfully used to maintain the security mechanisms.

Axiom	**Support**	**Remarks**
Acyclicity	Yes	Explicitly supported
Closure	Yes	It can be assumed that there are no types outside the type hierarchy
Rootedness	Yes	The type T_object is the root of the hierarchy
Pointedness	No	No support for pointedness
Supertypes	Yes	
Supertype Lattice	Yes	
Nativeness	Yes	Explicitly supported
Inheritance	Yes	Supports full Inheritance
Interface	Yes	Native + Inherited

Table: 3.3.6.2 Axiomatic support for the proposed model

Table 3.3.6.2 summarizes the various features that the proposed security model supports. Clearly the proposed model supports almost all the features of the axiomatic model.

Chapter 4 The Windows NT Security Management System

The current security model for the Windows NT operating system is very powerful and has many features. The User Manager provided by Windows NT is the primary method for the provision of security maintenance. Unfortunately this tool does not offer several features which would make the security managers's task more intuitive. This thesis demonstrates a new technique to support the security on a Windows NT platform.

Our system supports at least the following features:

- A more intuitive user interface so the administrative errors are less likely to be problematic.
- Simplified security management on a Windows NT platform.
- Avoid unnecessary creation of users / groups and
- Avoid redundant granting / revoking privileges.

Security maintenance can be compared to Dynamic Schema Evolution as both the processes involves the management of the changes while the system (database in the case of schema evolution and operating system in the case of the Windows NT) is in operation. The axiomatic model provides a solution for the Dynamic Schema Evolution in the object-based system so the same model can be used in security maintenance.

The new security model SAM (Security Axiomatic Model) makes the system security administration much easier and more efficient. The proposed model is implemented in the Windows NT operating system without changes to the operating system. Since the model is based on dynamic schema evolution, it can be implemented even while the operating system is in operation (i.e. without shutting down the system). The components of the axiomatic model are viewed as Windows NT objects for better

understanding and easier maintenance. For example, the 'types' in the axiomatic model are viewed as 'users/groups' in the proposed model. Similarly 'properties' are viewed as 'privileges' and hence 'essential properties' and 'essential supertypes' becomes 'essential privileges' and 'essential super users/groups,' respectively, in the proposed model.

4.1 Architecture Overview

The SAM security management system contains three important components. They are the Windows NT layer, the axiomatic layer and the security management system interface. The interaction between the components is shown in Figure 4.1.1.

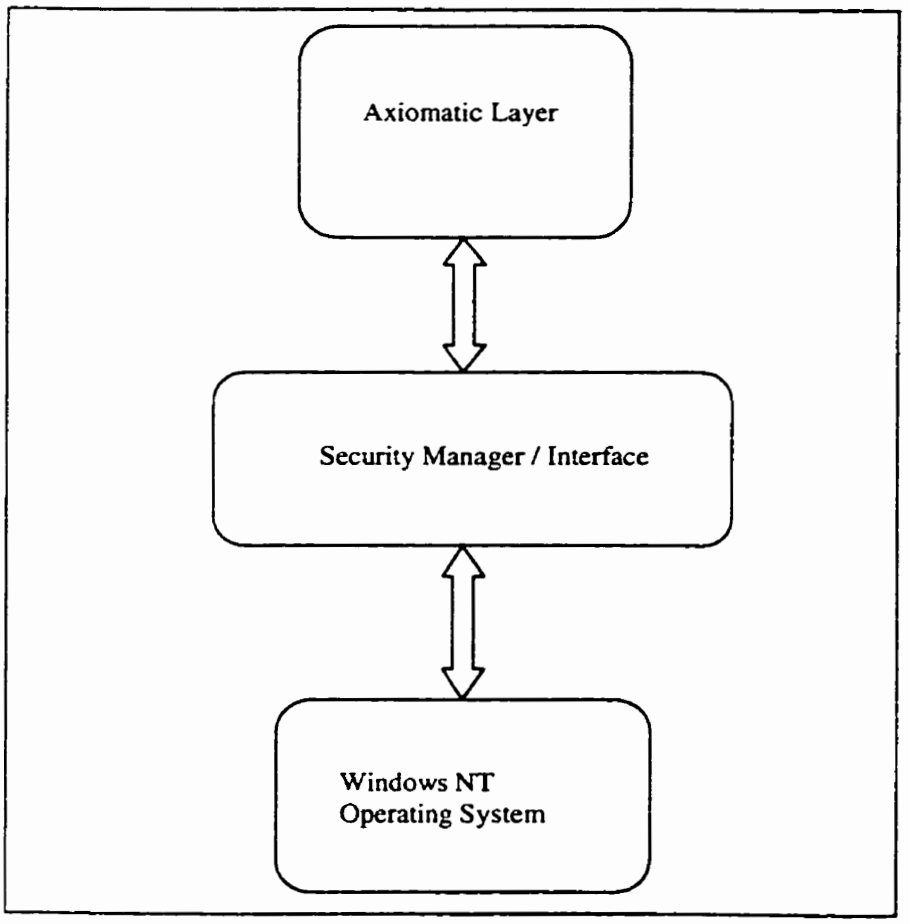


Figure 4.1.1 Different Layers of Security Management System

It can be seen that the security manager interface connects Windows NT's security system with the axiomatic model. Each component is discussed in the next few sections.

4.1.1 The Axiomatic Layer

This component stores all the Windows NT objects and their properties in terms of the axiomatic model. Windows NT's security model is a flat structure. Users cannot be subtypes of others nor can a group be a subtype of another group. However, the same flat structure can be expressed with an axiomatic model, thereby enabling both groups and users to be classified as types. The axiomatic model defines every type (users or groups) using subtypes and supertypes. SAM implements this object-oriented model with a file (external to NT's security system) that holds metadata about each of the types (users and groups). Therefore, the axiomatic layer is composed of two important components:

- The Axiomatic model
- The Metadata Axiom File (MAF)

NT user and group properties and their relationships are captured and expressed in terms of axiomatic properties. This model treats all the users and groups as types and privileges as properties. Any change in the subtype/supertype relationships or their properties result in corresponding change to the axiomatic model's properties. This means all the features supported by the axiomatic model such as: adding and dropping types, subtype, supertype and properties of a type are automatically supported by our security management system. Recall that Windows NT does not support these features because its security mechanism is not object-oriented.

The Metadata Axiom File is required because state information required by the axiomatic component is completely different from the structure of the traditional Windows NT security model. Thus the Metadata Axiom file holds all the data required to build the axiomatic model of the NT defined users and groups. Metadata stored in the MAF includes:

- All the types
- The subtypes of each type
- The supertype of each type
- Essential properties of all type

- Native properties contained in each type
- Inherited properties of all types

These properties play a key role in building the axiomatic model. This metadata provides the information required to efficiently manage security on a NT machine. The Security Manager stores these properties in the MAF so all information about user and group state is saved. The axiomatic model for the user is dynamic so each time our Security Manager is loaded, the axiomatic model is built from the metadata in the MAF, thereby restoring the exact state of each of the objects.

Although Windows NT does not support the features provided by the axiomatic model, the Security Manager provides an object-oriented security model. The Security Manager interface ensures that all changes made by our system are propagated to NT.

4.2 Windows NT security Mechanisms

The Windows NT's security model is flat structure and does not support any hierarchical structure, let alone an object-oriented one. NT supports their security model with the *User Manager*. The NT model's security features supported by the *User Manager* include:

- Add / Remove a Group
- Add / Remove a User
- Add /Remove a member of a group
- Add / Remove privileges of a group
- Add / Remove privileges of a User

Each of these functions is described in the sections below.

4.2.1 Add / Remove a group

NT's User Manager is used to add new groups to the system. Newly created groups do not have any privileges or user rights. A list of members can be added to the group who then inherit the corresponding rights and privileges. This means that each user must be

added to each group in which it should have privileges. It would be preferable to add groups of users to the newly created group thereby easing the process of creating new classes of users. In effect a hierarchy of user groups would be extremely helpful in security management.

Conversely, when a group is removed all members lose their membership. If the group has privileges, its members will all lose them unless they are explicitly given to the member through the granting of direct privilege. Ideally privileges could be grouped and formed into a hierarchy so that by dropping a subgroup the users would lose only a subset of privileges while maintaining those granted by the "super" group.

4.2.2 Add / Remove a User

New users initially belong to the "Users" group but the *User Manager* can insert them into additional groups. Users are atomic in that NT does not support the concept of a "user hierarchy" so users are inserted into groups only. When users are removed, they are physically removed from the system. The removed user is extracted from any groups to which they belonged. Finally, it should be noted that when a user is deleted it is completely removed from the system so even adding an identically named "new" user does not restore the old one.

4.2.3 Add / Remove Member to the group

Group membership can be specified while creating the group or user[1], or at any time after the group is created. Once a user is added to a group, all group privileges are inherited. Groups are composed of an arbitrary number of users but cannot contain other groups. In short, a group is only composed of existing users.

When a member is removed from a group it loses all privileges it inherited from the group except those that were granted directly. For example, if a user is given a privilege 'P' explicitly (direct privilege) which is also inherited from ones of its groups, the removal of the group does not remove 'P' because of the direct privilege.

[1] Users can only be inserted at the time they are created if the group has already been created.

4.2.4 Add / Remove Privilege of a group

Whenever a privilege is added to a group it is propagated automatically to all members. Unfortunately Windows NT does not provide a mechanism to view privileges inherited as a result of group membership. The *User Manager* only provides a list of direct user privileges. Therefore, changes to group privileges are not reflected when viewing the users in the *User Manager.* A list of users/groups with a particular privilege can be created but it is impossible to find a complete list of privileges held by a particular user.

Consider a scenario where we would like to remove one of a user's privileges. To accomplish this we would like to remove the direct privilege and the privilege from all groups to which the user belongs that have the privilege. Deleting the direct privilege is trivial and once completed the *User* Manager will correctly display the absence of the privilege. But what about the privileges inherited from the groups? NT is very robust in that deleting a privilege from a group will remove it from both the group and its members. Although this is correct, it is impossible to tell by simply looking at the user privileges if the privilege has been completely removed. If we identify all groups but one containing the privilege, it will remain and be undetectable.

4.2.5 Add / Remove privilege from a User

Users can have privileges granted to them from the *User Manager.* These are known as *direct privileges.* If this privilege is revoked the user will lose the direct privilege only. In other words, if the user is a member of a group that holds the privilege, the user will not lose it completely. This case is very difficult to handle with NT's *User* Manager as we discuss in the next section.

4.3 The Security Manger Interface

The security manager interface is a component that links the Windows NT security model and the axiomatic model. It ensures that all changes made by our system are propagated to NT. It provides a translation from/to Windows NT's security model and the axiomatic model. Since our model sees security privilege changes as schema evolution, updates to

NT privileges are propagated as axioms to the axiomatic model. Conversely changes within the Security Manager are propagated to NT as privilege changes.

4.4 System operation

SAM (Security Axiomatic Model) is installed above the native security model of Windows NT. The axiomatic model is represented by a directed graph where a node represents each user or group and the sub-type / super-type relationship is represented by an edge. An edge from a node A to node B indicates that the type A (user / group) is the supertype of type B. Further type B inherits all the properties of type A as expected.

For any two nodes N_i, N_j , if Ni is contained in the interface of Nj, then there must be a path between N_i and N_j. The interface of a particular node A is a set that contains all the nodes that are super-types of A.

Figure 4.4.1 is a snap shot of the object–oriented view of our security model for Windows NT. The type T_object is the root of all the other types (both users and groups). This means the type T_object has the minimum privilege and that all others inherit the privileges it possesses. The groups *Administrators*, *Power Users*, *Guests*, *Replicator* and *Users* are the immediate subtypes of T_Object. Since NT does not allow a group to be a member of another group, all groups are attached only to T_object, initially when the native security model is converted to the axiomatic model. The users *Db2Admin* and *Guest* are the members of the *Administrators* and *Guests* groups, respectively, so there is an edge from the type *Administrators* to the user *Db2Admin* and from *Guests* to *Guest*. The direction of property propagation is top-down. For example, the user *Db2admin* and groups *G_Group1* and *Administrator* inherit all the properties of *Administrators*. Similarly the users *guest* and *U_User1* inherit all the properties of the *Guests* group. To achieve greater clarity, different colors are given to System defined groups (Red), System defined users (Magenta), User defined groups (Green) and User defined Users (Yellow).

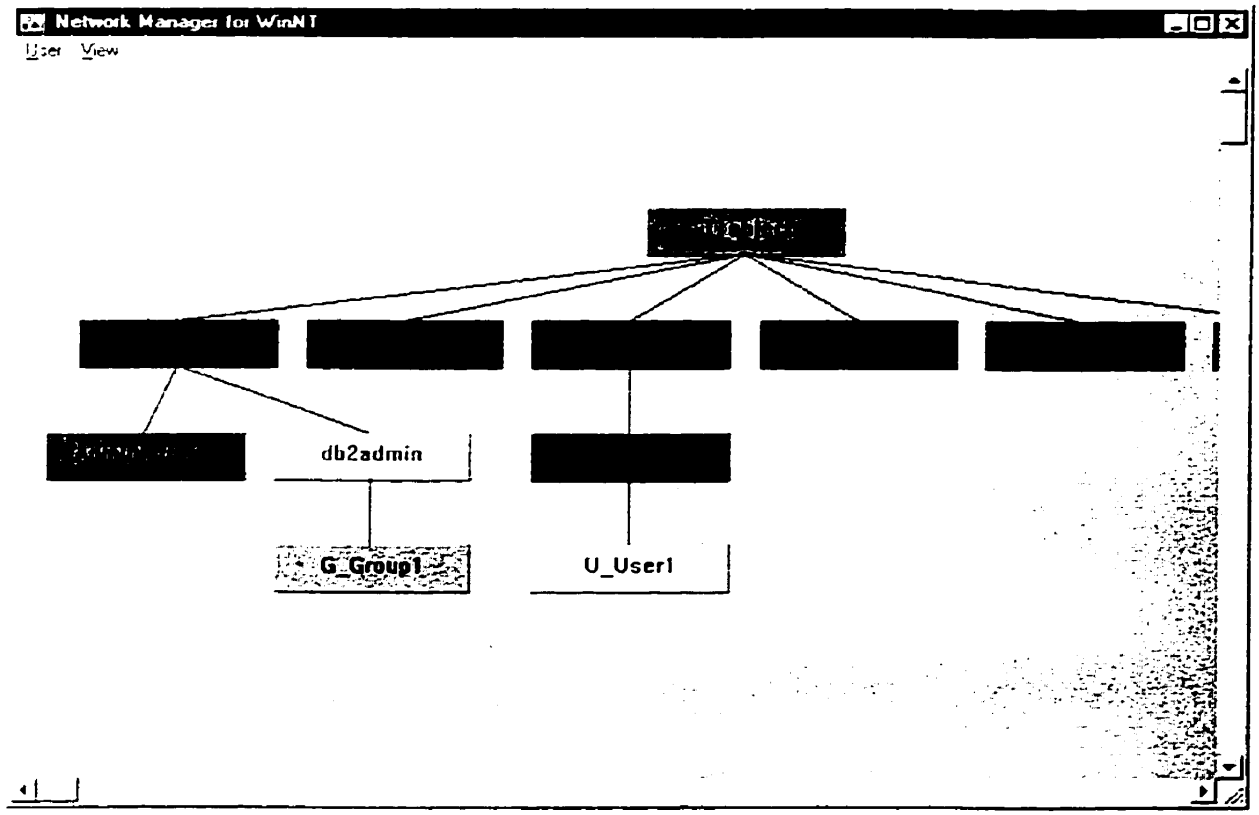


Figure 4.4.1 The Interface showing groups and users

We now summarize some of the axiomatic components in terms of Windows NT objects that were previously described in terms of axiomatic model:

Type: NT objects are commonly referred to as types. This includes both users and groups.

Properties: The privileges associated with each group or user are termed as properties.

Sub-Type and Super-Type: Subtyping permits an object to be built based on another. For example, when a user is added as a member of a group, then we say that the user is a subtype of group or the group is the supertype of the user. Both users and groups are viewed as types so a user can be a subtype of another user, group can be a subtype of another group or a group can be a subtype of another user. By subtyping, an object (user

or groups) inherits all the properties of the supertype. An object can have multiple supertypes and in that case it inherits the properties of all the supertypes.

Essential Super-Type: The essential Super-type of an object (user or group) contains all the users or groups that are essential to construct the object. All immediate super-types are essential and thus every group is an essential super-type to all its members.

Super-type Lattice: An object's type lattice contains the object and all its super-types.

We now turn to the security management features supported by the security manager:

- Add / Remove a Group
- Add / Remove a User
- Add /Remove a sub-type (both users and groups)
- Add / Remove privileges of a group
- Add / Remove privileges of a User

4.4.1 Add / Remove group

When a group is added, axiomatic metadata should be specified. This data might include the essential supertypes, immediate subtypes and the privileges the group possesses. Unlike NT's security model, SAM allows the users or groups to be essential supertypes or immediate subtypes. In other words, both the users and groups can contain others. The group and its privileges are sent to the axiomatic component, which adds this group as a new type. Once the axiomatic model is updated, the affected NT objects (users/groups/privileges) are modified.

The insertion process requires that the group is created and all supertype privileges are given to this group. Thus, the group has inherited all the properties of its supertype. These privileges must now be propagated to all its immediate subtypes. Two cases must be considered:

Case 1: If the immediate subtype is a user then the user is added to the group's membership roster. Once added to the group, NT propagates the group's privileges so there is no need to do this explicitly.

Case 2: If the immediate subtype is a group, then all of the group's privileges must be explicitly propagated to the subtype groups. This explicit propagation must be handled recursively to ensure that all children, grandchildren, *etc.* receive the necessary privileges.

Figure 4.4.1.1 (a) depicts groups *A*, *B*, *C*, *D*, *E* and *F* with a graphical depiction of the axiomatic model. Addition of group *X* as a supertype of *A* results in the propagation of *X*'s privileges *(P1, P2, P3, P4)* to the immediate subtype *A* and to groups *C*, *D*, *E* and *F* which is the behavior expected in an object-oriented inheritance hierarchy (see Figure 4.4.1.1 (b)).

Deleting a group removes the corresponding type from the axiomatic model. These changes are reflected to all its subtypes but the effect differs depending on the subtype's type (i.e. user or group). If the subtype is a user, it is removed from membership in this group. If it a group, the privilege is removed unless it is held as a direct privilege or inherited through another path. These changes are subsequently propagated recursively to all the affected types (groups and users) below this point in the hierarchy.

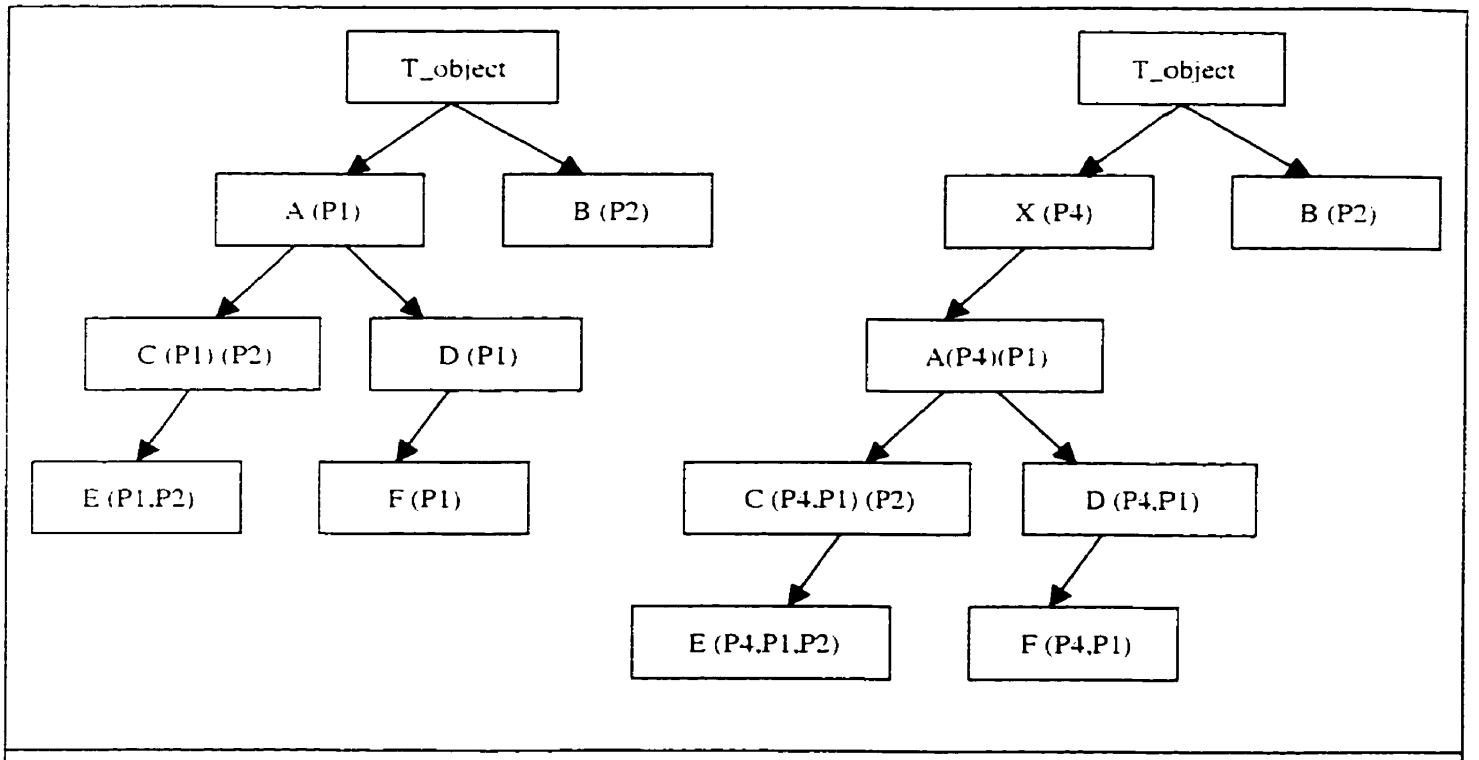


Figure 4.4.1.1: Addition of Groups (a) Before (b) After

4.4.2 Add / Remove a User

Adding a user to our system results in the creation of the new user on NT. The user can be created with no supertypes (except *T_object*) or as a subtype of a group or user. Once again several cases need to be considered:

Case 1: Users without supertypes are subtypes of *T_object*. This is consistent with the axiomatic model because every type must be a subtype of *T_Object*.

Case 2: Users added as a subtype of a group (or many groups) are made members of each supertype group. In this way the user inherits all the privileges of the supertype, which is consistent with the object-oriented model.

Case 3: Users added as a subtype of another user is not available under the native Windows NT security model. This feature would prevent the unnecessary creation of groups and make the NT security model easier to maintain. For an example let us consider *X* as a member of the group *profs* and *Y* member of the

group *students*. Now in the absence of *X*, *Y* has to do all the tasks that *X* performs (as *Y* is his research assistant). In other words *Y* needs all the privileges of *X*. This could be done using our model by just making *Y* as a subgroup of *X*. This also maintains the consistency of the group membership by making only professors as member of the group *profs*. and students of group *students*. At the same time there is no extra groups created or the same set of privileges granted. In the native NT model, the solution to this problem would be to create a new group, add all the privileges of *X* to that group and then add *Y* as the member of this newly created group. If the same operations were to be carried for many members of *profs* then this may result in the creation of a large number of groups making the maintenance of the security more difficult. This operation would permit a user to inherit all privileges of a particular user. Additional research would need to be undertaken to define the semantics of removing a supertype of a user, but the issue is beyond the scope of this thesis.

Our Privilege Propagation algorithm (see Algorithm 2) carries out this propagation for all cases.

If a user is deleted from the system, the axiomatic model is first updated and the corresponding effects are propagated to the NT objects. The Privilege Propagation algorithm (modified to revoke privilege) removes privilege from the object's subtypes. Finally, the user loses membership in the supertype's groups.

The algorithm used to add and remove groups and users is as follows:

Algorithm1: Addition or Removal of a group or user:

1. Update the axiomatic components including
   Essential Super-type
   Immediate sub-type
   Type Lattice

2. **If** X is a User **Then**
   Delete / Create X as User
   If the operation is addition **Then**
   Add X as the member of all its super-type
   Endif

Else

Delete / Create X as a new group
Revoke / Grant all the privileges of its super-types to X
Endif

3. Initialize the groupList with all the sub-types of X

4. **For each** element in the groupList **do**
If the element I is a user **Then**
Remove / Add this user I as a member to the group X
Else
revoke / grant the privilege of group X to this group I
Endif
For each sub-type of the group I **do**
Add this sub-type to the groupList
Endfor
Remove the element I from the list
If the groupList is empty **Then**
Return
Endif
Endfor
End Algorithm1

4.4.3 Modify sub-type relationship

Addition (removal) of an object (user or group) as a supertype of another object results in the addition (removal) of the set of essential supertype for the object too. Our model permits the supertype to be either a user or group. As in previous cases, once the axiomatic model is updated the process of privilege propagation and addition/deletion of users as members is performed on the NT machine iteself.

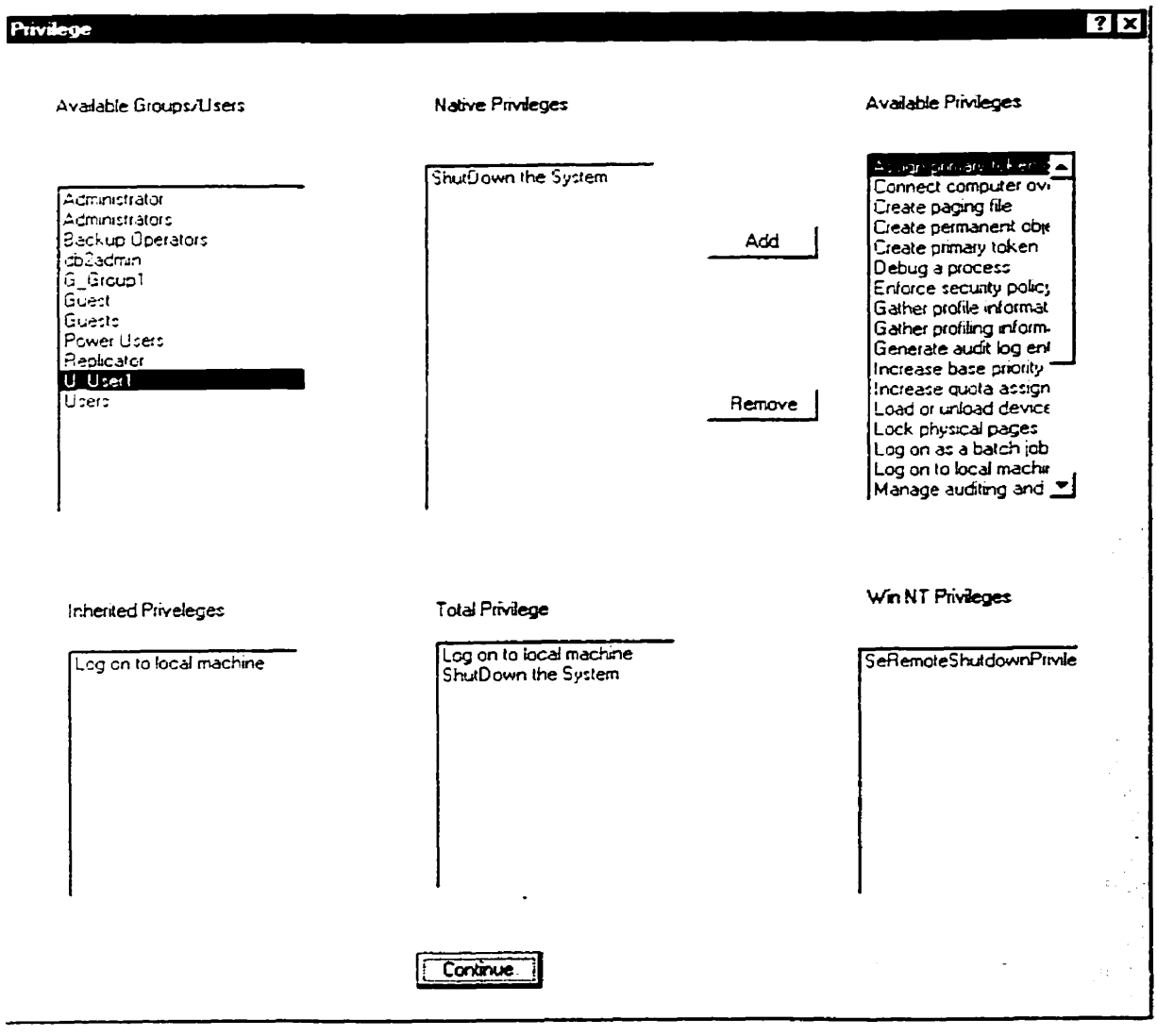


Figure 4.4.4.2: Interface Showing the Available Privileges

4.4.4 Add/Remove Privilege of a group

We now turn out attention to the specific issue of privilege management used in our system. Before presenting the details of our implementation we provide a few definitions[2]. Privileges in our model are broadly classified into three types:

- Native Privilege: are directly given to the types
- Inherited Privilege: are acquired by the types from the parent (user or group)

[2] These terms have been used intuitively early in the thesis but a more precise definition is required to describe this aspect of the implementation.

- Privilege Interface: the union of the inherited and the native privileges

Our system clearly presents a list of each privilege held by an object as illustrated in Figure 4.4.4.2. For example the figure depicts user *U_User1* membership in *Guests* and that he holds the 'Shutdown the System' privilege. The inherited privilege 'Log on to local machine' would not appear as an NT privilege (privileges as seen using the *User Manager*) because these are not displayed.

The addition and removal of privileges is the final aspect of our system to consider. The key issues here are related to how privileges are propagated throughout the object hierarchy and how these are passed onto the flat structure found in NT. A brief discussion of the addition and removal of privileges is provided followed immediately by a sketch of the algorithm that implements this aspect of the system.

Add a Privilege: When a privilege is added to a type, it is added to the set of native privileges. The privilege must be propagated to its sub-types. The process of privilege propagation is similar for both users and groups. These privileges only need to be granted to the groups because NT propagates them to the group's members on the systems behalf.

Remove a Privilege: Removing a privilege from a user or group requires it first be removed from the set of direct privileges. In our system, if the privilege is found in the set of inherited privileges it has been acquired from at least one of its parents. This means the privilege is not revoked from the Windows NT system and to do so would require that the corresponding privilege be removed from the parent. Alternatively, the object could be removed from the parent carrying the privilege (see Section 5.4.3). Once removed the privilege must be recursively applied to its subtypes.

The Privilege Propagation Algorithm is outlined below:

Algorithm 2: Privilege Propagation Algorithm

1. Add / Remove the privilege P from the set Native Privilege of type T
2. **If** privilege P not found in set Inherited Privilege of type T **Then**
   Add / Remove the privilege P from the type T (can be user or group)
   If T a group **Then**
   Add all the sub groups of the type T to the GroupList

```
        Else
            Add all the sub-types (both users and groups) of T to the GroupList
        Endif
        For each element I in the GroupList do
            Add / Remove the privilege P from the set Inherited Privilege
            If privilege P not found in set Direct Privilege Then
                Add / Remove the privilege P from the type T (can be user or
            group)
                If T a group Then
                    Add all the sub groups of the type T to the GroupList
                Else
                    Add all the sub-types (both users and groups)
                        of T to the GroupList
                Endif
            Endif
            Remove the element I from the list
            If the groupList is empty then return
        Endfor
EndAlgorithm 2
```

4.5 Summary

The system has been implemented and proven to provide an excellent intuitive interface that meets the primary goals of our project. The system is sufficiently flexible that a system administrator can use our system to deploy new roles, groups and users but if they choose to use the *User Manager* to complete a task, our system will resynchronize with those changes once it is restarted. In this way, both our system and the one provided by NT can be used for complementary tasks. Our model is much more intuitive and it should be further investigated with the ultimately goal of deploying it as native to NT.

Chapter 5 Conclusion And Future Directions

In this thesis a new security management model based on well-known schema evolution techniques in OBMSs is discussed. The model is successfully implemented on Windows NT above the original security model in such a way that it does not require modification or introduce conflicts to NT's current approach. One of the nicest features of this approach is that both the system and the *User Manager* can operate together. Any changes made with the *user manager* are reflected in our system and changes done using our tool can be seen with the *User Manager* in precisely the way you would expect.

The major motivation for this thesis are:

- The popularity of the axiomatic model among object-based systems in solving the problem of post-design modifications. The axiomatic model handles the problem of dynamic schema evolution effectively.
- The various security models used in different systems treats objects like groups and users as different entities. This increases the complexity of the system, as it requires redundant creation of groups, and granting and revoking of privileges. Thus a security model that treats both the users and groups as a single entity makes the process of security management much easier and more efficient.
- A new non-discretionary access control mechanism, has emerged called role based access control. According to RBAC rights and permissions are assigned to roles rather than individual users. Users acquire these rights and permissions by assigning membership to appropriate roles.
- The management of security is related to schema changes in different ways that were discussed in Chapter 3.

Thus a new security model was developed based on the axiomatic model and was successfully implemented on Windows NT machines.

Windows NT was chosen as a test platform because the various security operations like adding a group/user, removing a group/user, adding privilege to a group/user, removing a group from a group/user and modifying membership are similar to the schema changes adding a type, deleting a type, adding a behavior/property, deleting a behavior/ property and modifying subtype relationships, respectively. Also NT is one of the most common network operating system used in the market.

The *User Manager* on the other hand lacks many features. Some of the drawbacks associated with the Windows NT security model and its maintenance tool (the *User Manager*) addressed by this research includes:

- Lack of an object-oriented approach.
- Failure to provide a clear link between inherited privileges arising from participation in groups and the lack of a technique to extract this information in an easy clear way.
- Inherited privileges of the members are hidden.
- An improved visual interface to the security model

We have also demonstrated how our model permits the user to perform various operations that makes the management of security easier and more understandable. Some of the benefits of the new model include:

- Avoids unnecessary creation of groups.
- Prevents redundant and unnecessary granting and revoking of privileges.
- Provides a better visual interface that clearly illustrates privilege flow.
- Promotes object-oriented mechanism for maintaining the security model.
- Grants no more privilege than is necessary to perform a task. This property ensures the adherence to the security principle of least privilege.

Thus with the proposed model, the maintenance of security becomes much easier and more efficient. However there are some minor limitations and the following features will be added in future:

- ❖ Since the proposed model and the operating system should operate together, a lot of unnecessary processing is done to check whether the objects are updated by the Windows NT tool (the *User Manager).* If this redundant processing is avoided the speed of the system can be increased.
- ❖ Currently the model supports the maintenance of security objects like users and groups. This could be extended and be made to maintain other Windows NT objects like files and directories.
- ❖ The model supports complete inheritance. That is all the properties of a type both native and inherited are acquired by its subtypes. Another set of properties known as the user properties can be included which are specific to the user to which it is granted but are not propagated to its subtypes. By adding this property a user can possess some advanced privileges but still can have subtypes that do not have this privilege. This privilege set is helpful to a group/user that needs to inherit most but not all the privileges of its super user.

In summary our model makes the process of security management much easier and efficient. The model is object-oriented and a tree structured interface is provided that makes the model better than the *User Manager* – the tool that comes with the Windows NT. The model can be extended to other systems (like UNIX) by just modifying the layer that maps the system's objects (users, groups and privileges) into the axiomatic model's components (types and properties).

References:

[1] Peters, R.J, and Özsu, M.T. Axiomatization of Dynamic Schema Evolution in Objectbases, In 11th International Conference on Data Engineering, Taiwan, March 1995, pages 156 - 164.

[2] Özsu, M.T, Peters, R.J, Irani, B., Lipka, A., Munoz, A., and Szafron, D. TIGUKAT Object Management System: Initial Design and Current Directions, In Proceedings of CASCON'93, Toronto, Canada, October 1993, pages 595-611.

[3] Peters, R.J. TIGUKAT: A Uniform Behavioral Objectbase Management System. Ph.D thesis. University of Alberta. TR94 – 06. April 1994

[4] Ferrialo, D., and Kuhn, R. Role Based Access Control. *15th National Computer Security Conference.* 1992 pages 13 - 16.

[5] Ferrialo, D., Cugini, A., and Kuhn, R. Role Based Access Control : Features and Motivation . *11th Annual Computer Security Application Conference.* IEEE Computer Society Press, 1995 pages 127 - 131.

[6] Barkley, J. Implementing Role Based Access Control Using Object Technology. *First ACM Workshop on Role Based Access Control.* November 1995 pages 293 - 298.

[7] Barkley, J. Comparing Simple role Based Access Control Models and Access Control Lists. *Second ACM Workshop on Role Based Access Control.* August 1997 pages 127 - 132.

[8] Barkley, J, and Cincotta, A. Managing Role/Permission Relationships Using Object Access Types. *Third ACM Workshop on Role Based Access Control.* July 1998 pages 73 - 80.

[9] Hua, L., and Osborn, S. Modeling UNIX access control with a Role Graph. *In the Proceedings of the Ninth International Conference on Computing and Information.* June 1998, pages 131 –138.

[10] Minasi, M., Anderson, C., Creegan, E. Mastering Windows NT Server. Fourth Edition. 1997.

[11] Okuntseff, N. Windows NT Security, R&D Books. 1997.

[12] Elmasri, R., and Navathe, S.B. Fundamentals of Database Systems. Second Edition. 1994.

[13] Dally G., Buhrmaster, G., and Campbell, M. NT Security in open Academic Environment. *Second Large Installation System Administration of Windows NT Conference.* July 1999 pages 59 - 68.

[14] Smith, C.L., Coyne, E., Youman, C., and Ganta, S. A Marketing Survey of Civil Federal Government Organizations to Determine the Need for RBAC Security Product. SETA Corporation. 1996.

[15] Kim, W., and Lochovsky, F.H. Object-Oriented Concepts, Databases, and Applications. ACM Press.1989.

[16] Butterworth, P., Otis, A., and Stein J. The Gemstone Object Database Management System. *Communications of the ACM*, Vol.34 No. 10. October 1991, pages 64 - 76.

[17] Deux, O. The O_2 System. *Communications of the ACM*, Vol.34 No. 10. October 1991 pages 34 - 48.