DESIGN AND IMPLEMENTATION OF A HYBRID AND CONFIGURABLE ACCESS CONTROL MODEL

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

UĞUR TURAN

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

SEPTEMBER 2009

Approval of the thesis:

DESIGN AND IMPLEMENTATION OF A HYBRID AND CONFIGURABLE ACCESS CONTROL MODEL

submitted by **Uğur Turan** in partial fulfillments of the requirements for the degree of **Master of Science in Computer Engineering, Middle East Technical University** by,

Prof. Dr. Canan Özgen ______________________
Dean, Graduate School of **Natural and Applied Sciences**

Prof. Dr. Müslim Bozyiğit ______________________
Head of Department, **Computer Engineering**

Dr. Attila Özgit ______________________
Supervisor, **Department of Computer Engineering, METU**

Examining Committee Members:

Prof. Dr. M. Ufuk Çağlayan ______________________
Department of Computer Engineering, Boğaziçi University

Dr. Attila Özgit ______________________
Department of Computer Engineering, METU

Assoc. Prof. Dr. Halit Oğuztüzün ______________________
Department of Computer Engineering, METU

Dr. Cevat Şener ______________________
Department of Computer Engineering, METU

M.Sc. Mert Özarar ______________________
Department of Computer Technology and Information Systems, Bilkent University

Date: September 11, 2009

ABSTRACT

DESIGN AND IMPLEMENTATION OF A HYBRID
AND CONFIGURABLE ACCESS CONTROL MODEL

Turan, Uğur
M.S., Department of Computer Engineering
Supervisor : Dr. Attila Özgit

September 2009, 85 pages

A hybrid and configurable access control model is designed to satisfy the requirements of using different access control models in the same schema. The idea is arised to completely combine and configure the two main access control models, discretionary and mandatory which have been widely used in many systems so far with their advantages and disadvantages. The motivation originates from the fact that; in real life usage, discretionary based systems needs some strict policies and mandatory based systems needs some flexibility. The model is designed to combine these two appoaches in a single and configurable model, with some required real life extensions, in a conflict-free fashion and configurable degree of combination. Implementation of the model has been done and main important cases which shows the power and expressiveness of the model are designed and implemented. The authorization process is in the responsibility of the model which can be combined with secured authentication and auditing schemas. The new approaches as Role-Based, Context-Based and Temporal access control can easily be embedded in the model due to its generic and modular design.

Keywords: Access Control, Discretionary, Mandatory, Hybrid Access Control, Configurable Access Control

ÖZ

HİBRİT ve AYARLANABİLİR ERİŞİM KONTROL MODELİ
TASARIMI VE UYGULAMASI

Turan, Uğur
Yüksek Lisans, Bilgisayar Mühendisliği Bölümü
Tez Yöneticisi : Dr. Attila Özgit

Eylül 2009, 85 sayfa

Bir hibrit ve ayarlanabilir erişim kontrol modeli olan bu model, değişik erişim kontrol modellerini aynı şema altında kullanma gereksinimini karşılamak amacıyla tasarlanmıştır. Bu şekilde bir yazılım tasarlama fikri; avantajları ve dezavantajları ile birçok sistemde yaygınca kullanılan isteğe bağlı ve zorunlu erişim kontrol modellerini tamamen birleştirme ve ayarlanabilir hale getirme düşüncesinden ortaya çıkmıştır. Gerçek hayattaki kullanımlarda zorunlu erişim kontrol modellerinin bazı durumlarda esnekliğe izin verme, isteğe bağlı olanlarının ise bazı durumlarda sıkı kurallar uygulama ihtiyacı, bu motivasyonu ortaya çıkaran gerçekler olmuştur. Bu yazılım; bu iki erişim kontrol yaklaşımını, ayarlanabilir tek bir modelde bazı açılımlar tanımlayarak istenilen ölçüde çelişkilerden arındırarak birleştirmek üzerine tasarlanmıştır. Modelin kodlama aşaması tamamlanmış ve modelin gücü ve etkisini gösteren durumlar tasarlanmış ve kodlanmıştır. Yetkilerdirme aşamasından sorumlu olan model, güvenli kimlik doğrulama ve kayıt alma şemaları ile entegre edilebilir özelliğe sahiptir. Rol tabanlı, durum tabanlı ve zaman ayarlı erişim kontrol yaklaşımları, modelin genelgeçer ve modüler tasarımı sayesinde kolay bir şekilde yazılıma eklenebileceklerdir.

Anahtar Kelimeler: Erişim Kontrol, İsteğe Bağlı, Zorunlu, Hibrit Erişim Kontrol, Ayarlanabilir Erişim Kontrol

To My Father

ACKNOWLEDGEMENTS

I would like to thank to my supervisor Dr. Attila Özgit for his guidance, criticism and encouragements throughout the research.

I would also thank Mert Özarar for his suggestions.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

Access control has been a popular subject in software arena since the usage of digital resources which have been thought to have a degree of confidentiality, has increased in probability and easiness by the improvement of technology and the popularity of information sharing in digital environments. The increasing demand on the internetworking and related concepts brings the confidentiality problem with their widely usage as the degree of information sharing should have been controlled by defining limitations and restrictions on specified criteria. These criteria might depend on an operation, a user, a role or even time in order to apply some rules.

The main purpose of access control is to define limitations or restrictions to a user for the actions and operations that can be applied upon a digital resource [24]. Many different access control models have been proposed to satisfy different needs and criteria for different problem domains. Basically, all access control models have been based on two main perspectives, namely discretionary and mandatory. Discretionary Access Control (DAC) means that the rules specifying what is allowed or not, should be defined and controlled by the owner or originator of digital resource. Access control model of the first operating systems has been mainly based on this idea in order to give user a full controlled space to be used individually [12]. On the other hand, Mandatory Access Control (MAC) specifies that a central authority should define the rules [20]. In MAC, it is prohibited for users to use initiative for the rules since the access control policy has been predefined among the system. In addition, Role-Based Access Control (RBAC) can be stated as the third main access control paradigm, yet the idea behind this subject have been evolved by the fact that rules should be stated for the roles and user can be mapped to the roles in a dynamic and limited manner according to some criteria [23]. RBAC can be configured as having mandatory or discretionary properties for its rules [18]. Briefly, these two main access control structures can be used in coordination with other criteria as time, domain, roles etc.

Nowadays, the need for access control is strictly increasing and huge problem domains are evolved to enforce access control regulations. DAC assumes the domain as personal, giving the complete control to the owner of resource; while, MAC assumes the domain as military, assuring the complete dependence of the users to the rules proposed by the authority of the system. As problem domains are getting larger and intersecting with each other, the requirement for a flexible Access Control Model (ACM) satisfying discretionary and mandatory needs together, has been arisen. Enterprise domains are getting larger since digital information should be shared among different departments. Workers might have personal information in the domain and they might want to share them with other coworkers. However for the sake of security, other coworkers should not be from a different department as there should not be a probability of sharing of enterprise information in this way by a misusing worker. On the other way, military domains might need for exceptional situations in access control model for the urgent cases, as commander might want to send an urgent message for a lower ranking soldier, which should be hidden from the others. These ideas yield to a hybrid and configurable ACM in which existing ACM's can be applied in configurable extent and contradictions or security leaks which has a probability to emerge after this combination should have been assured to non-exist.

1.1. Description and Objectives of the Study

In this study, a new hybrid and configurable ACM is defined to satisfy multi perspective requirements of problem domains. The model is based on discretionary and mandatory access control perspectives and builds a configurable base for the composition. The main motivation behind this study is to be able to define exceptional cases for MAC and authority-based limitation for DAC. Different concepts like blacklists in access control are introduced to increase the adaption of the system to real world requirements. As a result, the composition of existing access control approaches makes this new hybrid model with the configurable structure.

The system is designed to be configured according to problem requirements. Additionally, the degree of composition of mandatory and discretionary rules and even their precedence is also made configurable. Consequently, the model fixes the problems in domains where one-way approaches cannot be fully adequate. The system is designed as a new authorization model as it is available to be combined with powerful authentication and auditing mechanisms to serve as a trusted base for all problem areas [24]. Besides, the configurable structure of the model brings the advantage of modularity and ability to extend the model with new concepts.

1.2. Organization of the Thesis

The thesis manuscript continues with Chapter 2, containing the description of the related work on access control mechanisms and especially with the studies on merging discretionary and mandatory access control models. Afterwards Chapter 3 is about the design of the formal model that is presented as both mathematical and pseudo code definitions. Chapter 4 is completely about the implementation details of the software and database. Finally the thesis is completed by a conclusion chapter and future work, which consists of the planned functionalities and paradigms to be embedded into the model.

CHAPTER 2

RELATED WORK

2.1. Related Work on Access Control

Access control paradigm is mainly based on two entities; namely, object and subject [20]. Object can be defined as entities upon which have a probability to be acted an operation. On the other hand, subject can be defined as entities that can access to an object in order to accomplish an operation. Hence, access control is mainly the control of the actions of subjects upon objects. The control should include accessibility and authorization and also act as a guard between subjects and objects. The main challenge in access control paradigm is the interchangeable role of entities between being a subject and object. For instance, if a user executes a program then the user acts as a subject and program is the object. After then, let the program reads a file. For this operation the program has become the subject and the target file is the object. This makes the definition of rules more complex and hard to maintain. In addition to that, granularity of subjects and objects is another main issue in access control paradigm. In the example above, the portion of the program code (even a thread), which tries to read the file, can be manipulated as a subject and the portion of file that is tried to be read can be treated as object. The granularity of subject and object definitions is a matter of choice in access control design, and complexity of time and space should be considered while setting the degree of granularity.

The main approaches of access control are mainly classified into two categories: discretionary and mandatory. Discretionary perspective simply gives the responsibility to the owner of the object whereas mandatory approach prefers a system authority based model [20]. Moreover, a third access control approach, RBAC states role-user relationship depending on some criteria and for the role-object right mapping either discretionary or mandatory approach might be selected [18, 23]. Many different approaches, taking different criteria such as time, content, context etc. are still exist, nonetheless these criteria forms an additional layer on mandatory or discretionary controls.

The only choice for an ACM is not only between discretionary or mandatory approach. The rights of the subject upon objects can be programmatically stored as Access Control Matrix (ACMX), Access Control List (ACL) or Capability List (CL). Lampson has first defined this concept on the basis of operating systems [16]. This model is refined by Graham and Denning as taking the complexity criteria into account [11]. The formalization of the model has been done by Harrison, Ruzzo, and Ullmann (HRU model) and this formalization led to concrete complexity and efficiency analysis of the model [12]. ACMX takes each subject as row and each object as column, storing the rights of a subject for an object in a cell. This is a wide approach however it has probability to be a sparse matrix. ACL is the column wise storage of ACMX for each object. The advantage of this approach is to have less storage in comparison to a sparse ACMX. However, it becomes harder to get a subject's rights in this implementation. The contra verse of this approach is the CL approach, with row wise storage for each subject. The preference among these three types should be application specific [24]. Due to complexity issues, Sandhu has shown that safety problem remains undecidable in general whereas Typed Access Matrix (TAM) model has been defined with unchangeable type of subjects and object to make the problem decidable [22]. This assumption defines a better way to prove security of a system nevertheless the strong typed structure should be taken into account in the design process of a system since it becomes harder to apply this argument to an existing system.

Another main preference for an access control approach is prioritizing integrity or confidentiality. This applies mainly to mandatory approaches where these two preferences generally named as Biba Model and Bell-LaPadula Model. In these models, the main operations are nothing but read and write. Biba states no read-down, no write-up principle for the integrity problem [7]. On the contrary, Bell-LaPadula states no read-up, no write-down principle to satisfy confidentiality requirements [2]. Those rules are dual in nature and this discussion brings out that a system should make a selection between these two models. There has been made a different modeling for the mandatory systems, namely lattice-based which builds clearance structure into a lattice with single root [21].

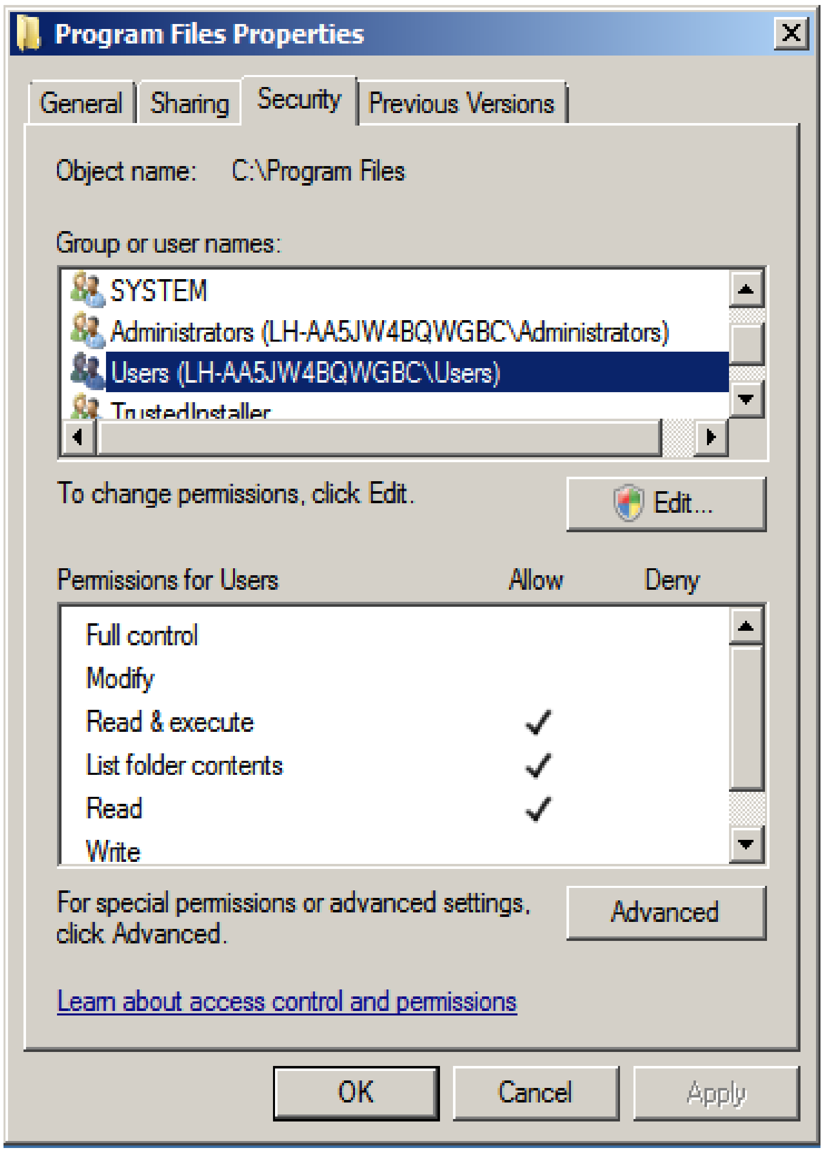
It is important to make a remark that access control model takes its place between authentication and authorization. The model should be applied to build a mechanism to serve as a Reference Monitor in cooperation with authentication and auditing mechanisms [24].

2.2. Access Control Issue in Operating Systems

General purpose and multi-user operating systems have mainly preferred DAC as the main access control mechanism. Because of being user-oriented system software, DAC gives users extent flexibility in order to manage their personal collections. However, system files and folders exist in the same file system and this coexistence evolves some security leaks while being used with DAC. To handle this issue, operating systems have tried to integrate some restrictions on DAC for the system resources.

Transitivity of user rights can be stated as a second security problem in operating systems. Subjects are only defined as user types; so executable programs are started and acted by the privileges of the user running it. This property makes the operating systems vulnerable to the Trojan Horses since a modified program might open doors to unwanted parties while user is unaware of happenings in background. Operating systems have been trying to solve these two issues over years and there are different partials solutions to these problems in different operating systems.

MS Windows family of operating systems are using access control lists for storing rules (Figure-2.1) and a DAC based access control mechanism is preferred mainly.


Program Files Properties

General | Sharing | Security | Previous Versions

Object name: C:\Program Files

Group or user names:

- SYSTEM
- Administrators (LH-AA5JW4BQWGBC\Administrators)
- Users (LH-AA5JW4BQWGBC\Users)
- TrustedInstaller

To change permissions, click Edit.

Permissions for Users	Allow	Deny
Full control		
Modify		
Read & execute	✓	
List folder contents	✓	
Read	✓	
Write		

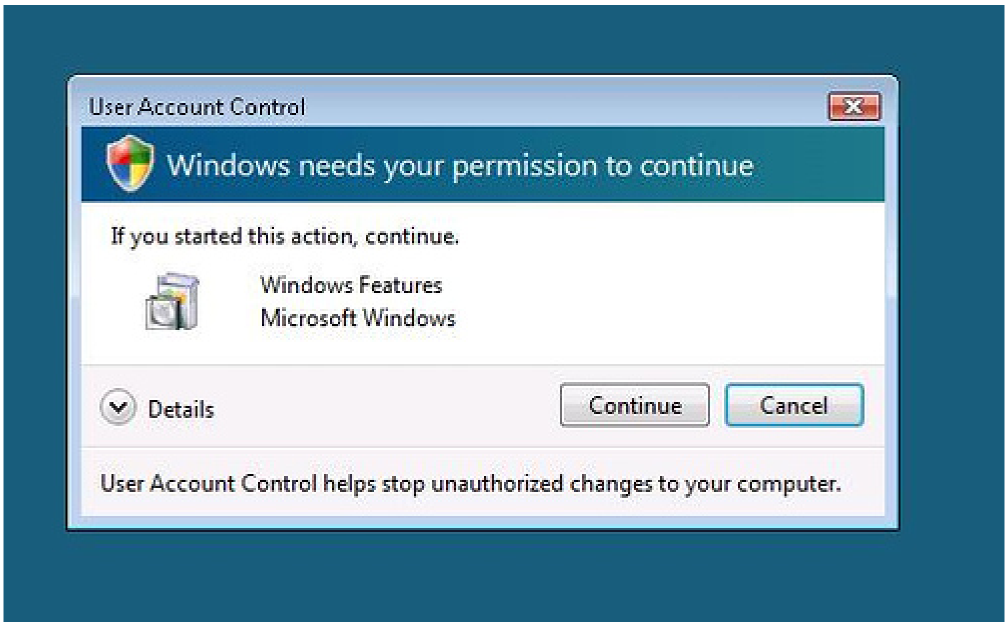
For special permissions or advanced settings, click Advanced.

Learn about access control and permissions

Figure 1.2.1: Windows ACL

There are two main user types, normal user and administrator. Administrator has full privilege over the system nevertheless normal user has a limited control area. Vista has offered User Account

Control (UAC) to start every user as standard user and grant administrator privilege if user is an administrator o can be authenticated as administrator. However if authentication is done as administrator, UAC works only as a pop-up question (Figure-2.2).



User Account Control

Windows needs your permission to continue

If you started this action, continue.

Windows Features
Microsoft Windows

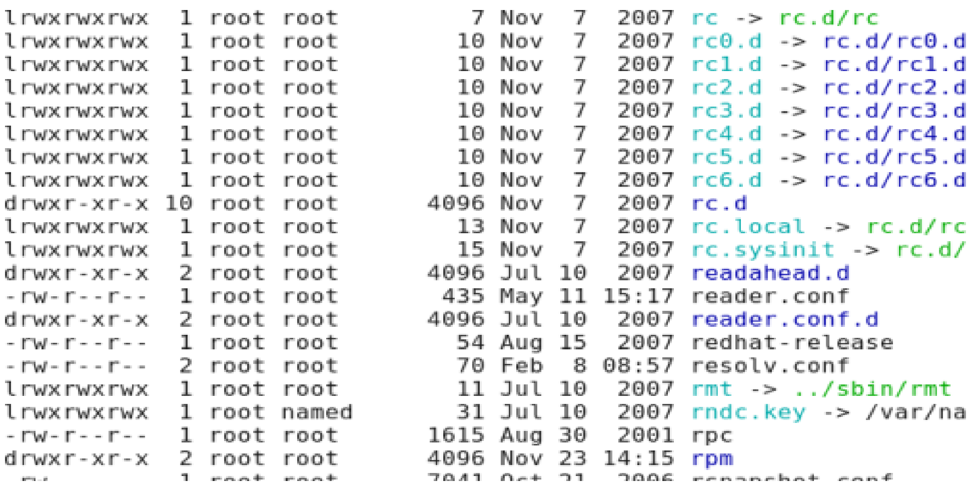
User Account Control helps stop unauthorized changes to your computer.

Figure 1.2.2: Windows UAC

Windows has a predefined user as System user in order to accomplish operating system based issues. Every modification is done to limit DAC with system regulations.

Linux based operating systems enforces DAC as 'r', 'w', 'x' attributes of objects for owner, group and other users. This is depicted in Figure-2.3.

This approach is the same as ACL and transitivity of user rights can be granted or rejected by a special attribute 's'. Linux has different user types as admin and normal as Windows, and user is asked admin authentication when needed. For traditional usage, initially providing user with admin authentication is not mainly preferred even by administrators in Linux.



```
lrwxrwxrwx  1 root root       7 Nov  7  2007 rc -> rc.d/rc
lrwxrwxrwx  1 root root      10 Nov  7  2007 rc0.d -> rc.d/rc0.d
lrwxrwxrwx  1 root root      10 Nov  7  2007 rc1.d -> rc.d/rc1.d
lrwxrwxrwx  1 root root      10 Nov  7  2007 rc2.d -> rc.d/rc2.d
lrwxrwxrwx  1 root root      10 Nov  7  2007 rc3.d -> rc.d/rc3.d
lrwxrwxrwx  1 root root      10 Nov  7  2007 rc4.d -> rc.d/rc4.d
lrwxrwxrwx  1 root root      10 Nov  7  2007 rc5.d -> rc.d/rc5.d
lrwxrwxrwx  1 root root      10 Nov  7  2007 rc6.d -> rc.d/rc6.d
drwxr-xr-x 10 root root    4096 Nov  7  2007 rc.d
lrwxrwxrwx  1 root root      13 Nov  7  2007 rc.local -> rc.d/rc
lrwxrwxrwx  1 root root      15 Nov  7  2007 rc.sysinit -> rc.d/
drwxr-xr-x  2 root root    4096 Jul 10  2007 readahead.d
-rw-r--r--  1 root root     435 May 11 15:17 reader.conf
drwxr-xr-x  2 root root    4096 Jul 10  2007 reader.conf.d
-rw-r--r--  1 root root      54 Aug 15  2007 redhat-release
-rw-r--r--  2 root root      70 Feb  8 08:57 resolv.conf
lrwxrwxrwx  1 root root      11 Jul 10  2007 rmt -> ../sbin/rmt
lrwxrwxrwx  1 root named     31 Jul 10  2007 rndc.key -> /var/na
-rw-r--r--  1 root root    1615 Aug 30  2001 rpc
drwxr-xr-x  2 root root    4096 Nov 23 14:15 rpm
```

Figure 1.2.3: Linux ACL

Linux has a special feature named SELINUX that enforces mandatory and role-based access control in addition to DAC. Security roles and classification levels are introduced to the Linux kernel and least required privilege has to be supplied for the kernel operations. The project is not compatible with some regularly used networking issues so it has not been popular for this compatibility problem. Generally, operating systems have a common perspective to the access control paradigm by offering and modulating DAC. Restrictions to the DAC for system objects are added however they could not turn to MAC totally since personalization is in high degree in their usage.

2.3. Related Work on Hybrid Systems and Motivating Factors

The need for combining mandatory and discretionary approaches has been realized staring with even initial systems. Early works on the target subject have dealt with the operating system security and tried to limit discretionary user space by mandatory regulations, using file name and extensions [8, 14]. Brewer and Nash has developed a security policy named "Chinese Wall" which gives discretionary dataset selection freedom to users among a conflict of interest group as these selections are used to clarify mandatory rules [9]. Since the policy has focused on commercial security, total configurability that also takes place in this study is restricted in the policy. Actually, "Chinese Wall" is not the start point of this perspective; Walter has described an information flow schema for operating systems. In this schema, processes has rights on an object according to previous objects it has been granted permission [27]. Another work has been completed by McColumn where associating every ACL with the object and granting of a right is done according to the intersection of associated ACL's [17]. Restrictions has been introduced and propagated with the object in the system that is more limited than pure discretionary systems. The perspective has dealt mainly with information flow security. The idea is close to this study except the interaction of different ACL's can also be configured in the process of granting access and the originating restrictions is not totally discretionary.

Another work has been done to handle multi-policy access control models in the same system [3]. However, it has been assumed that the object is managed with only one policy, which is not a combination actually; it can be regarded as different policies are living in the same system with clear borders among them. Besides, works on overriding discretionary access control has been done by defining not only granting or rejecting accesses but concentrating on also possible rights [19, 25]. The need is basically the same; improving mandatory with flexibility and limiting discretionary with bordered flexibility; however the improving and limiting should be managed and controlled as well. Bertino has presented exception based access control mechanism to manage the possible rights and relate the exceptions with the context [4]. The target was object-oriented systems and the exceptions say the last word in access control after the policy. The idea is a kind of base for this study and the motivation is getting the idea broader and configurable.

Earlier studies on hybrid access control mechanisms present a base and background for this study. As stated above, the general approach is limiting discretionary or making mandatory flexible. They have satisfied these issues according to the needed extent for their domain nevertheless many multi-domain systems need a fully configurable model. In addition to that, it should be clearly stated that the subject allowing the limitation or flexibility should also be controlled and limited for the extent of the restriction or exceptions discussed above. This will lead to a multiple root hierarchy where all domains can reside in their spaces and another normal or administrative subject should control every right including change the nature of a right by adding limitation or flexibility. This perspective could be reasoning for modern operating systems where there is not a clear distinction between DAC in user space and MAC in system space which is the optimal access control mechanism in operating systems.

This manuscript continues with the design (Chapter 3), in which the perspective of the proposed access control model is going to be revealed. Mathematical model and pseudo code procedure details are clearly stated and in addition to that remarks and discussions are made for the durability and integrity of the access control model.

CHAPTER 3

DESIGN OF THE MODEL WITH FORMAL AND PSEUDO CODE DEFINITIONS

Design concepts, mathematical description of access control model and pseudo code descriptions of the procedures are defined and described in this chapter of the thesis. Set and Relation based modeling are chosen for the completeness of our formal model. The pseudo code descriptions of the procedures are described as a practical proof of concept of the model. On the other hand, the design of the model should be carefully examined for the remarks and statements that are going to build a basis for future additions that might be intended for the extensions to this study.

3.1. Design Architecture

The design architecture of the proposed model represented in Figure-3.1 has been built in order to satisfy the needs of the model presented in previous chapter. The model is mainly composed of subjects, compartments and objects. Subject can be utilizer or owner of a compartment. Every compartment has a different access control schema in order to satisfy the needs of multi-domain environments. The operations can be divided into two as object operations and compartment operations. Compartment operations exist for management issues whereas object operations are the ones on which access should be controlled.

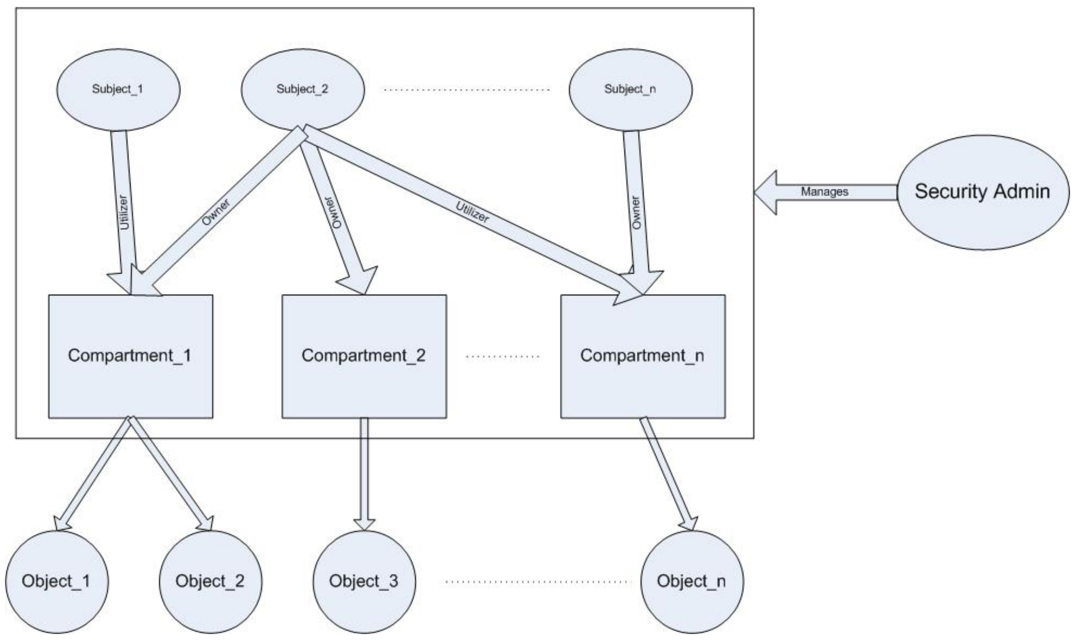


Figure 3.1.1: Design Architecture

If we map objects to files, then compartments mean more than folders since it has utilizers, owner and management rules. Subjects can be a combination of single real users such as person, process etc. (defined as actors in Formal Model), in order to satisfy possible security concurrency needs in future. A subject should be at least a utilizer of the compartment in order to access objects in that compartment. Every utilizer has a mandatory security level and every object has a security level and discretionary set. When this fact is combined with independent access control schema of different compartments, multiple access control models can reside together in the model.

The model is not designed as a single root hierarchy as security admin is only responsible for creation of compartments and owners compartment operation rights. Security Admin has no access for the objects. All objects in a compartment are managed mainly by the owner and secondly by utilizers. This hierarchy has been done in order to avoid vulnerabilities of single root hierarchies in trust mechanism. Security admin is able to limit owner's compartment operation rights and is able to determine which compartment operation rights can be shared by other utilizers. Some compartment operations are only owner specific. In addition to that some object operations are owner specific, too. Object operations can be expressed in combination of basic operations and all access control rules are for basic operations. In order for an operation to be granted to a subject, all basic operations building the operation should be granted. The operations and basic operations are only textual namings with no induced meaning for the model, as the model should be categorized as an external reference monitor.

Object creation is another newly approached issue in the model. Every utilizer has default security settings which are to be inherited by the objects created. The security settings are both for mandatory and discretionary. The ability of extending or reducing mandatory or discretionary defaults is another compartment operation right given by the owner.

The model has also a blacklist structure to guarantee the access control in future. In access control models there is an existing vulnerability such that the subject's access to an object is prohibited in the first place; nevertheless by changing the mandatory security level of a subject, the access might be falsely granted in the future. In order to prohibit the access completely for the future including all modifications, there should be added a blacklist entry for the basic operation. Blacklist entries have highest priorities to deny the access.

Finally, the model satisfies the needs of mandatory exceptions and discretionary limitations by satisfying both properties in the same compartment. Furthermore, multi-compartment model serves for multi-domain environments where all operations, objects and rights are totally different in compartments for a subject.

3.2. Formal Model

This section is dedicated to formal description of our proposed access control model. The descriptions consist of textual and formal definitions of the entities in the model and relations among entities are also defined using both ways. The terms “system” and “model” are used interchangeably throughout the manuscript.

Definition 0: ***Alphabet***

Alphabet is a set of letters, which constructs the words used by entity nomenclature in the system.

Alphabet = { a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0 }

Alphabet is described as a set in order to configure local preferences easily.

Definition 1: ***Language***

Language is a set of words, which are used by entity nomenclature in the system.

$$Language = (a + b + c + d + e + f + g + h + i + j + k + l + m + n + o + p + q + r + s + t + u + v + w + x + y + z + 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 0)^{+}$$

The nomenclature policy of the system is left to the implementation preferences.

Definition 2: ***Subject***

A Subject is an active entity, which is capable of accessing to objects. In the model, subjects refer to real and unique users of the system.

Definition 3: ***Object***

An Object is a passive entity, which can be accessed by subjects. In the model, there is a clear distinction between the terms object and subject according to their types. If an entity is to be treated as both subject and object in a system using the model, it should be defined for both classes to the system independently.

Definition 4: *Security_Admin*

"Security Admin" is a special subject that is responsible for administrative issues and/or operations in the system. The role and capabilities of a security admin are clearly defined in the end of this section.

Definition 5: *Basis_Subject_Set*

Basis_Subject_Set is simply the set of unary subjects in the system.

$$Basis\_Subject\_Set = \{\, x \mid x \text{ is a } Subject \,\}$$

Definition 6: *Actor_Set*

This set is spanning Basis_Subject_Set and includes the subjects constructed by one or more unary subjects. The elements of this set are mentioned as "Actor" hereafter. An actor that has n elements (where n ≥ 2) is treated as a different entity from the subjects it is composed of.

$$Actor\_Set = \{\, x \mid x \subseteq Basis\_Subject\_Set \,\}$$

The "Actor" definition builds a compatible base for the solution of problems that need to treat multiple subjects as a single entity. The concept allows a subject, having completely different rights than other subjects all of which composing the same actor.

Definition 7: *Object_Set*

Object_Set is the set of objects in the system.

$$Object\ \_Set = \{\, x \mid x \text{ is an } Object \,\}$$

Object is the only source in this model for which any access operation is granted or rejected.

Definition 8: *Compartment*

Compartment is a tuple, containing the set of element objects and its alias for nomenclature.

$$Compartment = (\{ o \mid o \in Object\_Set \}, n \in Language)$$

Compartments constitute clear borders for different domains in the model. They are different from directories in ordinary file system, as it will be seen that they have their own access control logic.

Definition 9: *Compartment_Set*

Compartment_Set is the set of compartments in the system.

$$Compartment\_Set = \{ x \mid x \text{ is } Compartment \}$$

Property 0: There cannot be two or more compartments with the same name.

$$\forall cs_1 \forall cs_2 \forall cn_1 (((cs_1,cn_1) \in Compartment\ \_Set \wedge (cs_2,cn_1) \in Compartment\_Set) \Rightarrow (cs_1 = cs_2))$$

Property 1: An object cannot be an element of more than one different compartment.

$$\forall o \forall c_1 \forall c_2 ((o \in Object\_Set \wedge c1 \in Compartment\ \_Set \wedge \exists cs \exists cn (o \in cs \wedge c_1 = (cs,cn)) \wedge c_2 \in Compartment\_Set \wedge \exists cs \exists cn (o \in cs \wedge c_2 = (cs,cn))) \Rightarrow (c_1 = c_2))$$

This property makes the model a multi-domain environment where different object from different domains can reside.

Definition 10: *Compartment_Ownership*

Every compartment has an owner, which is an Actor.

$$Compartment\_Ownership = \{ (c,s) \mid c \in Compartment\_Set \wedge s \in Actor\_Set\}$$

The ownership, as opposed to being a “utilizer” of a compartment for an Actor, gives some responsibilities for the compartment to the owner Actor, which is defined soon in this section.

Property 2: A compartment should have only one owner.

$$\forall c\, \forall s_1\, \forall s_2\, (\, (\, (c,s_1) \in Compartment\_Ownership \wedge (c,s_2) \in Compartment\_Ownership\,) \Rightarrow (s_1 = s_2)\,)\, \wedge \forall c\, (\, c \in Compartment\_Set \Rightarrow \exists s\, (\, (c,s) \in Compartment\_Ownership\,)\,)$$

The property should be carefully examined that the only owner of compartment is an actor that can be composed of many subjects according to the definition of Actor.

Definition 11: *Compartment_Utilizers*

Every compartment has a set of Actors, which are compartment_utilizers (i.e. "utilizer") of the compartment.

$$Compartment\_Utilizers = \{\, (c,s) \mid c \in Compartment\_Set \wedge s = \{\, z \mid z \in Actor\_Set\,\}\}$$

This utilizer relationship can be described as the ability to request for right for an operation to an object in the compartment.

Property 3: If an object is the owner of a compartment then the object cannot be the utilizer of the same compartment.

$$\forall o\, \forall c\, ((c,o) \in Compartment\_Ownership \Rightarrow \exists x\, ((c,x) \in Compartment\_Utilizers \wedge o \notin x\,)\,)$$

Definition 12: *Status_Set*

Actors, objects and compartments can be disabled or enabled. This can be used as a step in access control procedure.

$$Status\_Set = \{ (x,y) \mid (x \in Object\_Set \lor x \in Actor\_Set \lor x \in Compartment\_Set) \land y \in \{enabled, disabled\} \}$$

Property 4: An actor, compartment or object can have only one referring element in Status_Set.

$$\forall x\ \forall y\ \forall z\ (((x,y) \in Status\_Set \land (x,z) \in Status\_Set) \Rightarrow (z = y))) \land \forall x\ ((x \in Object\_Set \lor x \in Actor\_Set \lor x \in Compartment\_Set) \Rightarrow (\exists z\ ((x,z) \in Status\_Set))$$

Property 5: All enabled objects should be element of a compartment.

$$\forall o\ ((o \in Object\_Set \land (o,enabled) \in (Status\_Set) \Rightarrow (\exists c\ (c \in Compartment\_Set \land \exists cs \exists cn\ (o \in cs \land c = (cs,cn))))$$

The model has no physical object deletion procedure as deleting an object means that making it disabled without being element of a compartment. As a result, all enabled objects or objects that can be enabled, should be element of a compartment.

Definition 13: *Security_Level_Set*

Every compartment has a set of defined security levels with their nonnegative integer value and alias.

$$Security\_Level\_Set = \{ (c,l) \mid c \in Compartment\_Set \land l = \{ (t,w) \mid t \in N \land w \in Language\} \}$$

The entries for a compartment in the security level set refer to mandatory security levels in the system. The nonnegative integer field in these levels is designed to help building a dynamic level hierarchy in the implementation phase.

Property 6: There cannot be two or more security level defined for a compartment, with the same integer value or same name.

$$\forall c\, \forall l\, \forall x\, \forall y\, \forall z\, \forall w\, (((c,l) \in Security\_Level\_Set \wedge (x,y) \in l \wedge (z,w) \in l) \Rightarrow (x \neq z \wedge y \neq w))$$

Property 7: There cannot be two or more security level set defined for a compartment.

$$\forall c\, \forall l_1\, \forall l_2\, (((c,l_1) \in Security\_Level\_Set \wedge (c,l_2) \in Security\_Level\_Set) \Rightarrow (l_1 = l_2))$$

Definition 14: *Actor_Security_Level*

Every actor has a compartment-specific security level for the compartments for which the actor is utilizer or owner of.

$$Actor\_Security\_Level = \{ (c,s,l) \mid ((c,s) \in Compartment\_Ownership \vee \exists x\, ((c,x) \in Compartment\_Utilizers \wedge s \in x)) \wedge \exists w((c,w) \in Security\_Level\_Set \wedge l \in w) \}$$

Property 8: There should be only one security level for an actor, among the security levels defined for the compartment for which the actor is utilizer or owner of.

$$\forall c\, \forall s\, \forall l_1\, \forall l_2\, (((c,s,l_1) \in Actor\_Security\_Level \wedge (c,s,l_2) \in Actor\_Security\_Level) \Rightarrow (l_1 = l_2)) \wedge \forall c\, \forall s\, (((c,s) \in Compartment\_Ownership \vee \exists x\, ((c,x) \in Compartment\_Utilizers \wedge s \in x)) \Rightarrow \exists l\, ((c,s,l) \in Actor\_Security\_Level))$$

Property 9: Only the owner actor of a compartment can have 0 as integer value for its security level in a compartment. This can be regarded as the maximum security level in access control function definition, which will be defined soon.

$$\forall c\,\forall s\,(\,(\,(c,s) \in Compartment\_Ownership\,) \Rightarrow \exists l\ \exists k\,(\,(c,s,l) \in Actor\_Security\_Level \wedge l = (0, k)\,)\,) \wedge \forall c\,\forall s(\,(\exists x\,((c,x) \in Compartment\_Utilizers \wedge s \in x\,)) \Rightarrow \exists l\ \exists m\,\exists k\,(\,(c,s,l) \in Actor\_Security\_Level \wedge l = (m, k) \wedge m \neq 0)\,)$$

Definition 15: ***Basic_Operations***

The set of basic operations for a compartment, which can be a part of operations, are acted by actors upon objects in the compartment.

$$Basic\_Operations = \{\,(c,bop) \mid c \in Compartment\_Set \wedge bop = \{\,n \mid n \in Language\,\}\,\}$$

Definition 16: ***Operations***

The set of operations for a compartment is defined as subset of the Basic_Operations set defined for that compartment.

$$Operations = \{\,(c,op) \mid c \in Compartment\_Set \wedge op = \{\,(x,y) \mid \exists w\,(\,w = \{\,z \mid (c,z) \in Basic\_Operations\,\} \wedge x \subseteq w \wedge y \in Language)\,\}\,\}$$

The one crucial point is basic operations are for the permission control nevertheless actions by actors will refer to the operations. This design approach inhibits contradictory issues in the system as granted operation ‘x’ consists of {a,b} nevertheless denied one ‘y’ consists of {b,c}. The question is the basic operation ‘b’ is permitted or not. In addition to that if the indivisibility of these operations ‘x’ or ‘y’ are guaranteed by the mechanism then they can also be defined as basic operations, as well.

Definition 17: ***Owner_Specific_Operations***

Owner_Specific_Operations are the set of operations that only owner actor can process to all objects in the same compartment.

Owner_Specific_Operations = { removeObject, changeAllPermission }

These rights clearly shows that owner actor in a compartment is a privileged actor responsible for all actions in the compartment.

Definition 18: ***Object_Security***

Every object in the compartment has a security level as mandatory security level for each basic operation defined for that compartment and set of actors allowed discretionally.

$$Object\_Security = \{ (o,b,d,m) \mid o \in Object\_Set \wedge \exists co\ (\exists cs \exists cn\ (o \in cs \wedge co = (cs,cn)) \wedge \exists bop\ ((co,bop) \in Basic\_Operations \wedge b \in bop)) \wedge d \subseteq \{ p \mid \exists c((\exists x\ ((c,x) \in Compartment\_Utilizers \wedge p \in x) \vee (c,p) \in Compartment\_Ownership) \wedge \exists cs \exists cn\ (o \in cs \wedge c = (cs,cn))) \} \wedge \exists c \exists w(c \in Compartment\_Set \wedge \exists cs \exists cn\ (o \in cs \wedge c = (cs,cn)) \wedge (c,w) \in Security\_Level\_Set \wedge m \in w) \}$$

Access Control List is preferred for the system instead of Access Control Matrix, since not all actors are completely related with all objects, an access control matrix for this model can be sparse.

Property 10: There should be only one element for an object and a basic operation in the Object_Security set.

$$\forall o \forall b_1 \forall d_1 \forall m_1 \forall d_2 \forall m_2 (((o,b,d_1,m_1) \in Object\_Security \wedge (o,b,d_2,m_2) \in Object\_Security) \Rightarrow ((d_1 = d_2) \wedge (m_1 = m_2))) \wedge \forall o \forall b\ ((o \in Object\_Set \wedge \exists co\ (\exists cs \exists cn\ (o \in cs \wedge co = (cs,cn)) \wedge \exists x\ ((co,x) \in Basic\_Operations \wedge b \in x))) \Rightarrow \exists m \exists d\ ((o,b,d,m) \in Object\_Security))$$

Definition 19: ***Actor_Compartment_Defaults***

It is the default security schema for the objects added to a given compartment by the utilizer actor. Owner actor needs no such default.

$$Actor\_Compartment\_Defaults = \{ (c,s,b,d,m) \mid \exists x\, ((c,x) \in Compartment\_Utilizership \land s \in x) \land \exists x\, ((c,x) \in Basic\_Operations \land b \in x) \land d \subseteq \{ p \mid \exists x\, ((c,x) \in Compartment\_Utilizership \land p \in x) \lor (c,p) \in Compartment\_Ownership \} \land \exists r\, ((c,r) \in Security\_Level\_Set \land m \in r) \}$$

Defaults of an actor restrict an actor for the new object created by her.

Property 11: There should be only one element defined for a utilizer actor in a given compartment, in Actor_Compartment_Defaults set.

$$\forall c\, \forall s\, \forall b\, \forall d_1\, \forall m_1\, \forall d_2\, \forall m_2\, ((((c,s,b,d_1,m_1) \in Actor\_Compartment\_Defaults \land (c,s,b,d_2,m_2) \in Actor\_Compartment\_Defaults) \Rightarrow ((d_1 = d_2) \land (m_1 = m_2))) \land \forall c\, \forall s\, ((\exists x\, ((c,x) \in Compartment\_Utilizers \land s \in x) \Rightarrow \forall b\, \exists d\, \exists m\, ((c,s,b,d,m) \in Actor\_Compartment\_Defaults))$$

Definition 20: ***Compartment_Operations***

It is the set of operations that are relevant to the compartment administrative issues.

$$Compartment\_Operations = \{addObject, extendDiscDefaults, reduceDiscDefaults, makeHigherMandDefaults, makeLowerMandDefaults\}$$

These operations are administrative or restrictive operations that can be acted in a compartment rather than the ordinary operations that are acted upon objects.

Definition 21: *Owner_Specific_Compartment_Operations*

It is the set of operations, which are relevant to the compartment administrative issues; nevertheless owner cannot give permission to other compartment utilizers to do these operations. These operations are given as a right to the owner actor of the compartment upon creation of the compartment.

Owner_Specific_Compartment_Operations = { addSecurityLevel, addUtilizerActor, removeUtilizerActor, changeUtilizersDefault, changeUtilizersSecurityLevel, giveUtilizersCompartmentOperationRight, cancelUtilizersCompartmentOperationRight }

This set defines the compartment operations, which are specific to the owner because of security issue.

Definition 22: *Compartment_Ownership_Restrictions*

It consists of three sets; one of which has elements from Compartment_Operations and defines the owner's Compartment_Operations rights, the second has elements from Compartment_Operations as well and defines what kind of Compartment_Operations can be given as a right to the other utilizers from the owner and the last defines which Owner_Specific_Compartment_Operations, owner actor has. This can be defined in the creation time of a compartment, owner changing process or modification of owner actor's rights done by the security admin. The ones in Owner_Specific_Compartment_Operations cannot be given to others, too.

$$Compartment\_Ownership\_Restrictions = \{ (c,s,z,t,d) \mid (c,s) \in Compartment\_Ownership \wedge z \subseteq CompartmentOperations \wedge t \subseteq CompartmentOperations \wedge d \subseteq$$

$$Owner\_Specific\_Compartment\_Operations \}$$

As it can be inferred, the utilizer actors are restricted by their permissions and defaults, while the owner actor is just restricted by his power. The degree of restriction is configurable which makes the model adaptive to all domains.

Property 12: If there are any CompartmentOperations, which the owner actor can give to other utilizers, then giveOthersCompartmentOperationRight should exists among owner actor's Owner_Specific_Compartment_Operations rights.

$$\forall c\, \forall s\, \forall z\, \forall t\, \forall d\, (((c,s) \in Compartment\_Ownership \land (c,s,z,t,d) \in Compartment\_Ownership\_Restrictions \land t \neq \emptyset) \Rightarrow (giveOthersCompartmentOperationRight \in d))$$

This property is actually for the cross control of the permissions.

Property 13: The set of Compartment_Operations which owner can give to utilizers should be a subset of the set that owner actor can perform.

$$\forall c\, \forall s\, \forall z\, \forall t\, \forall d\, (((c,s) \in Compartment\_Ownership \land (c,s,z,t,d) \in Compartment\_Ownership\_Restrictions \land t \neq \emptyset) \Rightarrow (t \subseteq z))$$

Property 14: There should be only one element, for all owner actors and for all compartments, in Compartment_Ownership_Restrictions set.

$$\forall c\, \forall s\, ((c,s) \in Compartment\_Ownership \Rightarrow \exists z\, \exists t\, \exists d((c,s,z,t,d) \in Compartment\_Ownership\_Restrictions)) \land \forall c\, \forall s\, \forall z_1\, \forall t_1\, \forall d_1\, \forall z_2\, \forall t_2\, \forall d_2\, (((c,s,z_1,t_1,d_1) \in Compartment\_Ownership\_Restrictions \land (c,s,z_2,t_2,d_2) \in Compartment\_Ownership\_Restrictions) \Rightarrow (z_1 = z_2 \land t_1 = t_2 \land d_1 = d_2))$$

Definition 23: ***Utilizer_Compartment_Operations***

It defines which utilizer in the compartment has what kind of Compartment_Operations rights.

$$Utilizer\_Compartment\_Operations = \{ (c,s,o) \mid (\exists x\ ((c,x) \in Compartment\_Utilizers \wedge s \in x)) \wedge o \subseteq Compartment\_Operations \}$$

Property 15: There should be only one element, for all utilizers and for all compartments, in Actor_Compartment_Operations set.

$$\forall c\, \forall s\, \forall o_1\, \forall o_2 (((c,s,o1) \in Utilizer\_Compartment\_Operations \wedge (c,s,o_2) \in Utilizer\_Compartment\_Operations) \Rightarrow (o_1 = o_2)) \wedge \forall c\ \forall s\ (\exists x\ ((c,x) \in Compartment\_Utilizers \wedge s \in x) \Rightarrow \exists z\ ((c,s,z) \in Utilizer\_Compartment\_Operations))$$

Property 16: The Compartment_Operations rights of utilizer actors can include a Compartment Operation not existing in the set which owner actor can give them because of a later modification by a special actor named ‘Security Admin’ as changing compartment owner or direct modification on owner actors rights, which will be defined later, on Compartment_Operations, the set of which can be given by the owner actor. However no utilizer can have more Compartment_Operation rights in comparison to owner.

$$\forall c\, \forall s_1\, \forall s_2\ \forall o\ \forall t\ \forall z\ \forall d\ ((\exists x\ ((c,x) \in Compartment\_Utilizers \wedge s_1 \in x) \wedge (c,s_1,o) \in Utilizer\_Compartment\_Operations \wedge (c,s_2) \in Compartment\_Ownership \wedge (c,s_2,z,t,d) \in Compartment\_Ownership\_Restrictions) \Rightarrow (o \subseteq z))$$

Definition 24: ***Operation_Blacklist***

It defines the blacklisted actors for basic operations of objects. The actors added need not to be a utilizer actor of same compartment which the object is belonging to. In addition to that the owner actor can be added to the blacklist.

$$Operation\_Blacklist = \{ (o, v, s) \mid o \in Object\_Set \land \exists co (\exists cs \exists cn (o \in cs \land co = (cs,cn)) \land \exists x ((c,x) \in Basic\_Operations \land v \in x)) \land s \in Actor\_Set \}$$

Definition 25: *Compartment_ACM_Schema*

It defines how access control check in a compartment will be done. The options are only mandatory, only discretionary, mandatory or discretionary, both mandatory and discretionary. This configuration will be the major modifiable step in access control method, which will be given as pseudo code later.

$$Compartment\_ACM\_Schema = \{ (c,s) \mid c \in Compartment\_Set \land s \in \{ M, D , D \lor M, D \land M\} \}$$

This schema defines the behavior of the compartment in the process of granting or denying an access right. In addition to that, having different schemas make compartments totally independent from each other.

Property 17: There should be only one element for all compartments in Compartment_ACM_Schema set.

$$\forall c \forall s1 \forall s2 (((c,s1) \in Compartment\_ACM\_Schema \land (c,s2) \in Compartment\_ACM\_Schema) \rightarrow (s1 = s2)) \land \forall c (c \text{ is } Compartment \rightarrow \exists s ((c,s) \in Compartment\_ACM\_Schema))$$

Definition 26: *Security_Admin_Operations*

It is the set of operations that can only be carried out by security admin. These operations are necessary for the administration of the ACM.

Security_Admin_Operations = { defineSubject, defineActor, createCompartment, deleteCompartment, setObjectEnabled, setObjectDisabled, setActorEnabled, setActorDisabled, setCompartmentEnabled, setCompartmentDisabled, addOperation, changeCompartmentOwner, removeActor, changeOwnersCompartmentOperationRestrictions, addToBlacklist, removeFromBlacklist, removeOperation, removeSubject }

Remark: Recall that, since Security_Admin is not an element of Actor_Set, these special actors do not have rights or process abilities that an ordinary actor possesses.

3.3 Pseudocode Definitions

These definitions are referring to the procedural logic of the model.

addObject

An actor is trying to add an object to a compartment with its discretionary set and mandatory security level for each basic operation defined for that compartment.

```
IF    the object is already in the compartment
      OR
      the actor is neither owner nor utilizer of that compartment
      OR
      the actor is disabled
      OR
      the compartment is disabled
      OR
      the security level given for any basic operation is not
      defined for that compartment
      OR
      any of the given basic operations is not defined for that
      compartment
      OR
      any of the actors in the discretionary set for all basic
      operations is neither utilizer nor owner of the compartment
      OR
      “addObject” does not exist among the compartment operation
      rights of that actor
      OR
      (the actor is not owner AND any of the discretionary sets
      for a basic operation contains an actor  which does not
      exist in the actor’s default discretionary set for that
      basic operation AND “extendDiscDefaults” does not exist
      among the compartment operation rights of that actor)
      OR
      (the actor is not owner
            AND
      any of the discretionary sets for a basic operation does not
      contain an actor that exists of the actor’s default
      discretionary set for that basic operation
            AND
      “reduceDiscDefaults” does not exist among the compartment
      operation rights of that actor
      )
      OR
      (the actor is not owner AND any of the mandatory security
      levels for a basic operation is lower in integer value
      -meaning a more secure level- than the actor’s default
      mandatory security level for that basic operation, comparing
      security levels by their integer value AND
      “makeLowerMandDefaults” does not exist among the compartment
      operation rights of that actor)
      OR
      (the actor is not owner AND any of the mandatory security
      levels for a basic operation is higher than the actor’s
      default mandatory security level for that basic operation,
      comparing security levels by their integer value AND
      “makeHigherMandDefaults” does not exist among the
      compartment operation rights of that actor)
THEN
```

```
      do nothing
ELSE
      define object to the system by adding to the object set
      add object to that compartment
      set status of the object as enabled
      add all discretionary sets and mandatory security levels for
            all basic operations for that object
ENDIF
```

Owner_Specific_Compartment_Operations

addSecurityLevel

Owner is trying to add new security level for the owned compartment.

```
IF    the actor is not owner of that compartment
      OR
      the actor is disabled
      OR
      the compartment is disabled
      OR
      there is a security level defined for that compartment with
      the same integer value or the same name with the new
      security level
      OR
      “addSecurityLevel” does not exist among compartment
      operation rights of the owner for that compartment
THEN
      do nothing
ELSE
      add the new security level for that compartment
ENDIF
```

addUtilizerActor

Owner is trying to add an actor as utilizer to the owned compartment.

```
IF    the actor which is trying to do the operation, is not owner
      of the compartment
      OR
      the actor to be added is already utilizer or owner of that
      compartment
      OR
      the owner is disabled
      OR
      the compartment is disabled
      OR
      the security level for the new utilizer is not defined for
      that compartment
      OR
      the integer field of the security level of the new utilizer
      is 0
      OR
      any of the mandatory security levels given for each basic
      operation as defaults is not defined for that compartment
      OR
      any of the actors in discretionary set given for each basic
      operation as defaults is neither utilizer nor owner of that
      compartment
      OR
      “addUtilizerActor” does not exist among compartment
      operation rights of the owner for that compartment
      OR
      any of the given basic operations in defaults is not defined
      for that compartment
      OR
      the compartment operation rights given for the new actor is
      not subset of the set of compartment operations which can be
      granted to utilizers by the owner
THEN
      do nothing
ELSE
      add the actor as utilizer to that compartment
      define the security level of the new actor for that
            compartment
      define the defaults of the new actor for each basic
            operation for that compartment
      define the set of allowed compartment operations for the new
            utilizer in that compartment
ENDIF
```

removeUtilizerActor

Owner is trying to remove utilizer actor from owned compartment.

```
IF    the actor which is trying  to do the operation, is not owner
```

```
      of the compartment
      OR
      the actor to be removed is not utilizer of that compartment
      OR
      the owner is disabled
      OR
      the compartment is disabled
      OR
      “removeUtilizerActor” does not exist among compartment
      operation rights of the owner for that compartment
THEN
      do nothing
ELSE
      remove the utilizer actors from compartment utilizers set
      remove the security level entry defined for the utilizer
            actor in that compartment
      remove the actor from all discretionary set entries for all
            basic operations defined for that compartment
      remove all compartment default entries for the actor in that
            compartment
      remove compartment operations entry for that actor in that
            compartment
ENDIF
```

changeUtilizersDefault

The owner is trying to change the defaults of a utilizer actor for a basic operation in owned compartment.

```
IF    the actor which is trying to do the operation, is not owner
      of the compartment
      OR
      the actor whose defaults to be changed for a basic operation
      is not utilizer of that compartment
      OR
      the owner is disabled
      OR
      the compartment is disabled
      OR
      the security level for the new default is not defined for
      that compartment
      OR
      any of the actors in discretionary set given for the new
      default is not utilizer or owner of that compartment
      OR
      “changeOthersDefault” does not exist among compartment
      operation rights of the owner for that compartment
      OR
      the given basic operation for the new default is not defined
      for that compartment
THEN
      do nothing
ELSE
      change default entry of the actor for that basic operation
            with the new default entry
ENDIF
```

changeUtilizersSecurityLevel

The owner is trying to change mandatory security level of a utilizer actor in an owned compartment.

```
IF    the actor which is trying to do the operation, is not owner
      of the compartment
      OR
      the actor whose security level to be changed is not utilizer
      of that compartment
      OR
      the owner is disabled
      OR
      the compartment is disabled
      OR
      the new security level of the utilizer is not defined for
      that compartment
      OR
      the integer field of the new security level is 0
      OR
      changeUtilizersSecurityLevel does not exist among
      compartment operation rights of the owner for that
      compartment
THEN
      do nothing
ELSE
      change the security level entry for the utilizer in that
            compartment
ENDIF
```

giveUtilizersCompartmentOperationRight

The owner is trying to give new compartment operation right to utilizer actor in that compartment.

```
IF    the actor which is trying to do the operation, is not owner
      of the compartment
      OR
      the actor to whom new compartment operation right to be
      given is not utilizer of that compartment
      OR
      the owner is disabled
      OR
      the compartment is disabled
      OR
      giveUtilizersCompartmentOperationRight does not exist among
      compartment operation rights of the owner for that
      compartment
      OR
      the new compartment operation does not exist among the set
      of compartment operations defined for the compartment
      OR
      the new compartment operation is not an element of the set
      of compartment operations which can be granted to other
      utilizers by the owner
      OR
      the new compartment operation is an element of the set of
      compartment operations that the utilizer already has right
THEN
      do nothing
ELSE
      add the compartment operation entry for the utilizer actor
            in that compartment
ENDIF
```

cancelUtilizersCompartmentOperationRight

The owner is trying to cancel the compartment operation right of a utilizer actor in that compartment.

```
IF    the actor which is trying to do the operation, is not owner
```

```
      of the compartment
      OR
      the actor to whom new compartment operation right to be
      given, is not utilizer of that compartment
      OR
      the owner is disabled
      OR
      the compartment is disabled
      OR
      cancelUtlizersCompartmentOperationRight does not exist among
      compartment operation rights of the owner for that
      compartment
      OR
      the compartment operation does not exist among the set of
      compartment operations defined for the compartment
      OR
      the compartment operation is not an element of the set of
      compartment operations that the utilizer already has right
THEN
      do nothing
ELSE
      remove the compartment operation entry for the utilizer
            actor in that compartment
ENDIF
```

Owner_Specific_Basic_Operations

changeAllPermissions

The owner is trying to change the security settings of the object for a basic operation in owned compartment.

```
IF    the actor trying to do the operation is not the owner of the
      compartment
      OR
      the object is not in that compartment
      OR
      the owner is disabled
      OR
      the compartment is disabled
      OR
      the object is disabled
      OR
      the new security level is not defined for that compartment
      OR
      the given basic operation is not defined for that
      compartment
      OR
      any of the actors in discretionary set given for the basic
      operation is not utilizer or owner of that compartment
THEN
      do nothing
ELSE
      change the object's mandatory and discretionary security
            entries for the basic operation in that compartment
ENDIF
```

removeObject

The owner is trying to remove the object from the owned compartment.

```
IF    the actor which is trying to do the operation, is not owner
      of the compartment
      OR
      the object is not in that compartment
      OR
      the owner is disabled
      OR
      the compartment is disabled
      OR
      the object is disabled

THEN
      do nothing
ELSE
      remove object from compartment
      change object's enable/disable status as disabled
      remove all discretionary and mandatory security entries of
            that object for all basic operations defined for that
            compartment
ENDIF
```

Access_Control_Method

hasRight

The actor is trying to get authorization for processing the operation on an object in same compartment

```
IF    the actor trying to do the operation, neither owner nor
      utilizer of the compartment
      OR
      the object is not in that compartment
      OR
      the actor is disabled
      OR
      the compartment is disabled
      OR
      the object is disabled
      OR
      the operation is not defined for that compartment
      OR
      the actor is blacklisted for the object for any of the basic
      operations which are elements of the operation tried to be
      processed
      OR
      (
      (the compartments ACM schema is 'M' AND the security level
      of the actor for that compartment is higher than the
      security level defined for the object in that compartment
      for all basic operations which are elements of the operation
      tried to be processed, comparing security levels by their
      integer value)
            OR
      (the compartments ACM schema is 'D' AND the actor is not an
      element of the discretionary set defined for the object in
      that compartment for all basic operations which are elements
      of the operation tried to be processed)
            OR
      (the compartments ACM schema is 'D∨M' AND ( the actor is not
      an element of the discretionary set defined for the object
      in that compartment for all basic operations which are
      elements of the operation tried to be processed AND the
      security level of the actor for that compartment is higher
      than the security level defined for the object in that
      compartment for all basic operations which are elements of
      the operation tried to be processed, comparing security
      levels by their integer value)
            OR
      (the compartments ACM schema is ‘D∧M’ AND (the actor is not
      an element of the discretionary set defined for the object
      in that compartment for all basic operations, which are
      elements of the operation tried to be OR the security level
      of the actor for that compartment is higher than the
      security level defined for the object in that compartment
      for all basic operations, which are, elements of the
      operation tried to be processed, comparing security levels
      by their integer value)
      )

THEN
      do nothing
ELSE
      grant permission
ENDIF
```

Security_Admin_Operations

addSubject

Security Admin is trying to add a new subject to the system.

```
IF    the subject already exists
THEN
      do nothing
ELSE
      add new subject to subject set
ENDIF
```

addActor

Security Admin is trying to add a new actor to the system.

```
IF    the actor already exists
      OR
      any of the composing subjects does not exist
THEN
```

```
      do nothing
ELSE
      add new actor to actor set
      make actor enabled
ENDIF
```

enableObject

Security Admin is trying to enable an object in a compartment.

```
IF    object is enabled
      OR
      object is not in any compartment
      OR
      object does not exist in the compartment
THEN
      do nothing
ELSE
      change object's status to enabled
ENDIF
```

disableObject

Security Admin is trying to disable an object in a compartment.

```
IF    object is disabled
      OR
      object does not exist in the compartment
THEN
      do nothing
ELSE
      change object's status to disabled
ENDIF
```

enableActor

Security Admin is trying to enable an actor.

```
IF    actor is enabled
THEN
      do nothing
ELSE
```

```
      change actor’s status to enabled
ENDIF
```

disableActor

Security Admin is trying to disable an actor.

```
IF    actor is disabled
THEN
      do nothing
ELSE
      change actor’s status to disabled
ENDIF
```

enableCompartment

Security Admin is trying to enable a compartment.

```
IF    compartment is enabled
THEN
      do nothing
ELSE
      change compartment’s status to enabled
ENDIF
```

disableCompartment

Security Admin is trying to disable a compartment.

```
IF    compartment is disabled
THEN
      do nothing
ELSE
      change compartment’s status to disabled
ENDIF
```

createCompartment

Security Admin is trying to create compartment.

```
IF    there exists a compartment with the same name
```

OR
owner name exists among utilizer names
OR
any of the security levels given for the new utilizers does not exist in given security level set for the new compartment
OR
any of the security levels given for the new utilizer's defaults for each basic operation does not exist in given security level set for the new compartment
OR
any of the basic operations given for the new utilizer's defaults does not exist in given basic operation set for the new compartment
OR
any of the basic operations given for the operations does not exist in given basic operation set for the new compartment
OR
any of names given for the utilizer's discretionary default does not exist as utilizer or owner name for the new compartment
OR
any of the compartment operations for the utilizers does not exist in compartment operation set
OR
any of the compartment operations for the owner does not exist in compartment operation set
OR
any of the compartment operations, which can be given by owner, does not exist in compartment operation set
OR
any of the owner specific compartment operations which owner has does not exist in owner specific compartment operation set
OR
owner can give any operation to utilizers nevertheless "giveUtilizersCompartmentOperationRight" does not exist in the owner's specific compartment operation set
OR
any of the security level's degree given for the new utilizers is negative
OR
the compartment operation set given for the new utilizers is

```
      not a subset of the compartment operation set which contains
      the compartment operations owner has right to do
      OR
      the compartment operation set given for the owner that
      contains the compartment operations owner can grant to
      utilizers, is not a subset of the compartment operation set
      which contains the compartment operations owner has right to
      do
      OR
      the given security level set for the new compartment
      contains elements with same name or integer value
      OR
      the given ACM schema for the new compartment is not defined
      in the system
THEN
      do nothing
ELSE
      add compartment to compartment set
      add compartment ownership entry for the new compartment and
            given owner actor
      add compartment utilizership entries for the new compartment
            and for all given utilizer actors
      add security level entries for the new compartment,
            including the one with integer value 0 for the owner
      add actor security level entries for the new compartment and
            for all given utilizer actors and given owner actor
      add basic operation entries for the new compartment
      add operation entries for the new compartment
      add actor compartment default entries for the new
            compartment for all given utilizer actors and for all
            basic operations
      add compartment ownership restrictions entry for the new
            compartment and for given owner
      add actor compartment operations entries for the new
            compartment for all given utilizer actors and given
            owner actor
      add ACM schema entry for the new compartment
      make compartment enabled
ENDIF
```

removeCompartment

Security Admin is trying to remove a compartment.

```
remove all security levels defined for that compartment
change all compartment objects’ status to disabled
remove compartment ownership entry for that compartment
remove all compartment utilizership entries for that compartment
remove enable/disable status of that compartment
remove all security level entries of utilizer and owner actors for
      that compartment
remove all basic operations defined for that compartment
remove all operations for that compartment
remove all object security entries for that compartment
remove all compartment default entries for all utilizer actors,
      defined for that compartment
remove compartment ownership restriction entry for the owner,
      defined for that compartment
remove all actor compartment operations entries for all utilizers,
      defined for that compartment
remove all blacklist entries defined for the objects in that
      compartment
remove ACM schema entry for that compartment
remove compartment from existing compartments
```

addOperation

Security Admin is trying to add new operation for a compartment.

```
IF    any basic operation constructing the operation is not
      defined for the compartment
      OR
      any operation defined for that compartment has the same name
      with that operation
THEN
      do nothing
ELSE
      add operation for that compartment
ENDIF
```

addToBlacklist

Security Admin is trying to add new blacklist entry for an actor, which is to be blacklisted for an object and for a basic operation that is defined for a compartment.

```
IF    basic operation is not defined for that compartment
      OR
      object is not in the compartment
THEN
      do nothing
ELSE
      add blacklist entry for that object and basic operation
ENDIF
```

removeFromBlacklist

Security Admin is trying to remove blacklist entry for an actor that exists for an object and for a basic operation, which is defined for a compartment.

```
IF    the blacklist entry for the object and basic operation does
      not exists
      OR
      basic operation is not defined for that compartment
      OR
      object is not in the compartment
THEN
      do nothing
ELSE
      remove blacklist entry for that object and basic operation
ENDIF
```

changeCompartmentOwner

Security Admin is trying to change owner of a compartment.

```
IF    new owner of the compartment is already owner of that
      compartment
THEN
      do nothing
ELSE
      IF    new owner of the compartment is already utilizer of
            that compartment
      THEN
            remove compartment utilizership of new owner for that
                  compartment
            remove security level entry of the new owner for that
```

```
                compartment
          remove the new owner from all discretionary sets for
                all basic operations and for all objects in that
                compartment
          remove compartment default entries for the new owner
                for that compartment
          remove compartment operation entries for the new owner
                for that compartment
      ENDIF

      change old owner’s compartment ownership entry with new
            owner for that compartment
      change old owner’s security level entry with new owner for
            that compartment
      change old owner’s compartment ownership restriction entry
            with new owner for that compartment
      change old owner’s compartment operation entry with new
            owner for that compartment
      change old owner with new owner in all discretionary sets
            for all basic operations and for all objects in that
            compartment
ENDIF
```

changeCompartmentOwnershipRestrictions

Security Admin is trying to change owner’s compartment restrictions for a compartment, which are the compartment operations, that owner has right to do, the compartment operations that owner has right to grant to a utilizer and the owner specific compartment operations that owner has right to do.

```
IF    the compartment operations which owner can grant to
      utilizers is not subset of the compartment operations which
      owner can do
      OR
      the compartment operations which any utilizer can do is not
      subset of the compartment operations which owner can do
THEN
      do nothing
ELSE
      IF    the compartment operations which owner can do is not
```

```
            empty set
            AND
            “giveUtilizersCompartmentOperationRight” does not
            exist among the compartment operations which owner can
            do
      THEN
            do nothing
      ELSE
            change owners compartment operation restrictions for
            that compartment
      ENDIF
ENDIF
```

removeActor

Security Admin is trying to remove actor from the system

```
IF    there exists a compartment which the actor is owner of
THEN
      do nothing
ELSE
      remove all blacklist entries for the actor
      remove all compartment operation rights for the actor
      remove all compartment default entries for the actor
      remove the actor from all discretionary sets of object
            security entries
      remove all actor security level entries for the actor
      remove enable/disable entry for the actor
      remove all compartment utilizership entries for the actor
ENDIF
```

removeOperation

Security Admin is trying to remove an operation for a compartment.

```
IF    the operation is not defined for that compartment
THEN
      do nothing
ELSE
      remove operation for that compartment
ENDIF
```

removeSubject

Security Admin is trying to remove a subject from the system.

```
IF    any actor in the system has the subject in its definition
THEN
      do nothing
ELSE
      remove subject from subject set
ENDIF
```

CHAPTER 4

IMPLEMENTATION OF THE MODEL SOFTWARE

4.1. Architectural Design

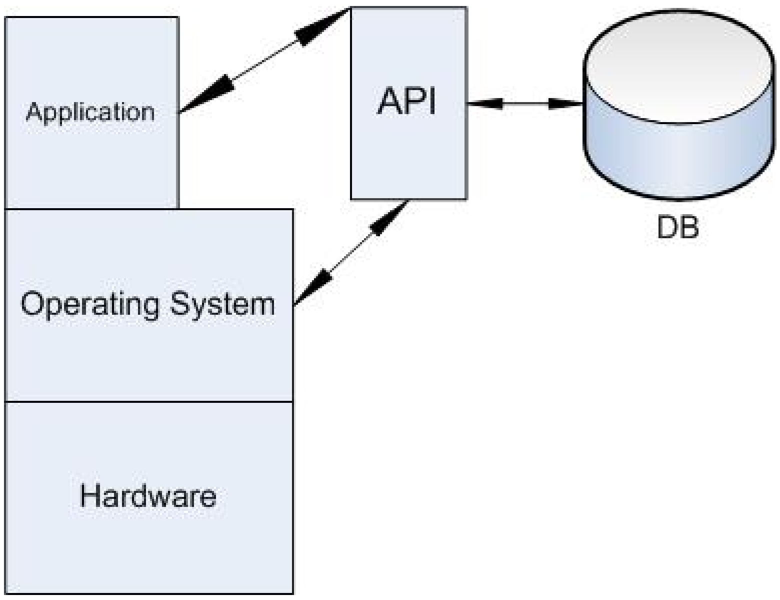


Figure 4.1.1: Model Architecture

The model architecture (Figure 1.1.1) is designed as an independent entity and interfaced by an Application Programming Interface (API). API includes the implementation of the procedures given in pseudo code definitions. The requesting entity of the access control mechanism could be either an application or operating system. Every operation, for which the request has come, has been defined to the model and the result should be performed according to the response the model gives. The API and DB can completely be thought as a black box responsible for authorization. According to this perspective, the model can be marked as a network unit, too.

The architecture is designed as, the integration of the API and requesting each operation to the API, are the responsibilities of the application.

4.2. Database(DB) Design

The entity-relationship diagram of the model (Figure 1.2.1) represents DB structure. To point out, there are many conditional actions in the model as seen in pseudo code definitions. There are two alternatives as assertions for these conditions, at DB layer or at code layer. Defining these assertions at DB layer might cause DB dependent behaviors, which decreases the independency of the design. Instead, code based high level assertions have been chosen to avoid this behavior.

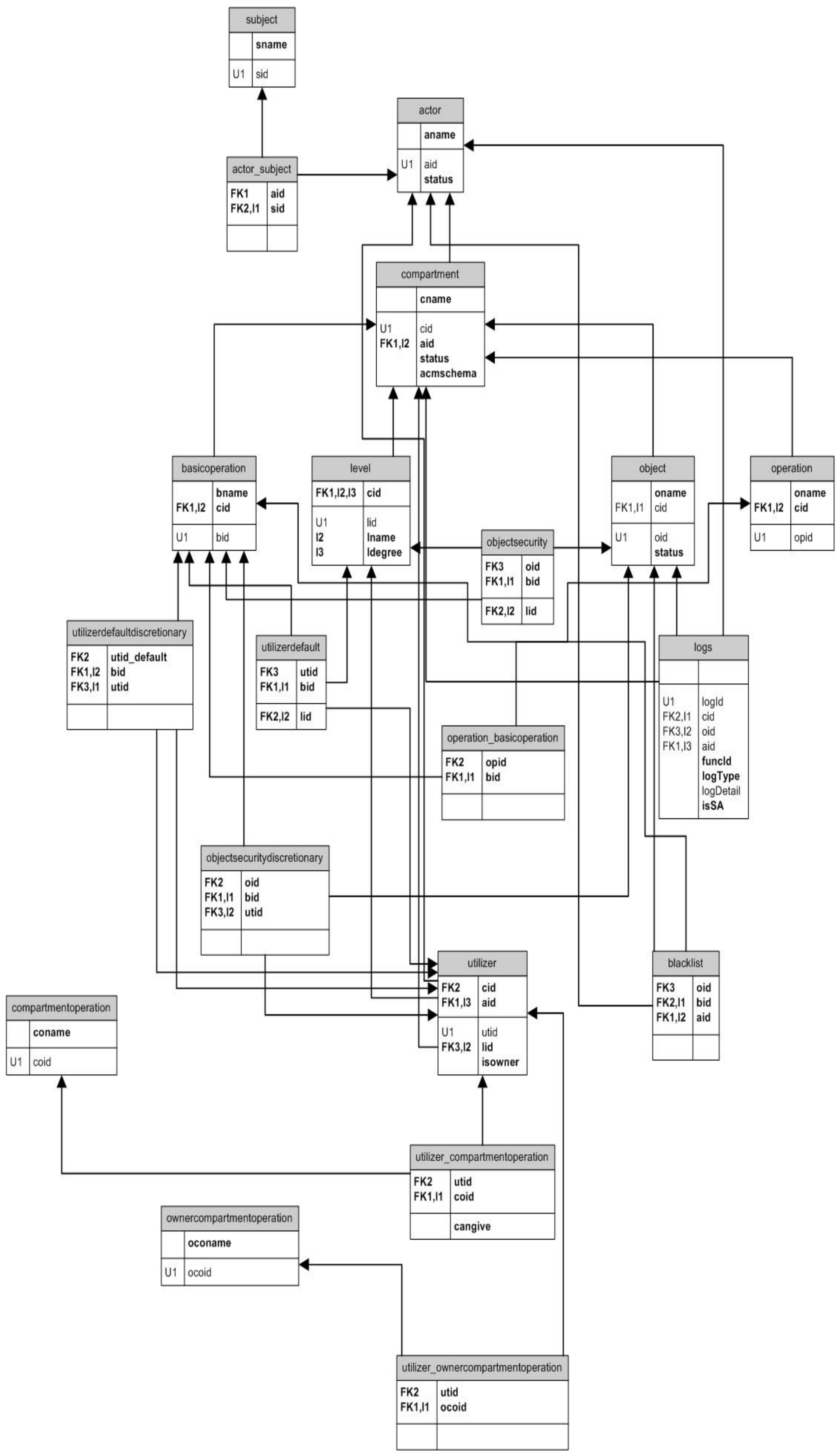


Figure 4.2.1: DB Entity Relationship Diagram

4.3. Analysis of Software Design Choices

The software should be investigated on three main criteria; efficiency, concurrency and durability. These are the main factors that can directly affect the performance of the software. Efficiency factor is critical for the real time systems, assuming that every operation should be intercepted by the API to grant or deny the operation. Nevertheless the software should not be investigated completely, since main functionality of the API lies on "hasRight" method. Other methods are administrative whereas hasRight will be in each operation of the outer architecture. The performance of the API is directly dependent with the DB performance. Among the implementation, the base of software has been built compatible with running as a web-service. There might be architectures, which will use API as a reference guard in between application and database instead of using as a library. On these architectures, an extra communication cost may be added to each inquiry.

Concurrency is the second criteria for the software as it has been designed to response to multiple requests in a synchronized fashion. The model has been implemented in a manner that all methods are atomic and all atomic requirements are single methods. The application using the API, should not define any extra methods, which are composed of at least two API methods. All methods in API has transaction rollback mechanism, any change destructing the integrity of the information in the database can be rolled back.

Durability can be seen as the last main criteria for the software since a failure in the system might be resulted as no operation can be carried out since the response of the API is the main prerequisite for every operation. The multiple cluster approach can be used in web service implementation to decrease the load between them. In addition to that, when API used as a library, the durability is going to be in responsibility of DB and main architecture.

4.4. Technology Choices

The main aim of the implementation phase of the study is to prove the concept based on the defined model Java is chosen as the programming language to utilize the advantages of a pure object oriented environment and Java 1.5 is selected as the application framework. An object-oriented analysis for high cohesion has been done for the processes in the model. The model is implemented as an API that can be used in any application for asking user rights. The modular design of the model makes the architecture compatible to serve as a web service.

Netbeans 5.5 is used as Integrated Development Environment (IDE) in development phase. MySQL 5.0 is selected as Database Management System (DBMS) and Hibernate 3 is used as Object-Relation Mapping (ORM) technology for persistency and database independent design. It is important to point out that the model is designed as a separate entity and the responsibility to ask the model for the permissions in each action, is completely of the application using the API. As a result, native libraries are no more needed since operations are only literals for the model as described above.

4.5. API Design

API design has been done by one to one correspondence with pseudo code definitions. All methods have been defined as static as the classes are defined stateless so there is no need for objects in API usage. The parameters are defined as String in order to satisfy the consistency with web services. Method names and parameters are listed in detail in Figure 4.5.1.

Method Summary	
static int	**addActor**(java.lang.String SAName, java.lang.String actorName, java.lang.String[ ] subjectNames)
static int	**addObject**(java.lang.String actorName, java.lang.String objName, java.lang.String compName, java.lang.String[ ] basicOps, java.lang.String[ ] level, java.lang.String[ ][ ] discSet)
static int	**addOperation**(java.lang.String SAName, java.lang.String compName, java.lang.String opName, java.lang.String[ ] basicOpNames)
static int	**addSecurityLevel**(java.lang.String ownerName, java.lang.String compName, java.lang.String levelName, int levelDegree)
static int	**addSubject**(java.lang.String SAName, java.lang.String subjectName)
static int	**addToBlacklist**(java.lang.String SAName, java.lang.String bopName, java.lang.String objName, java.lang.String compName, java.lang.String actorName)
static int	**addUtilizerActor**(java.lang.String ownerName, java.lang.String compName, java.lang.String actorName, java.lang.String levelName, java.lang.String[] basicOps, java.lang.String[] level, java.lang.String[ ][ ] discSet, java.lang.String[] compOps)
static int	**cancelUtilizersCompartmentOperationRight**(java.lang.String o wnerName, java.lang.String compName, java.lang.String utilName, java.lang.String coOpName)
static int	**changeAllPermissions**(java.lang.String ownerName, java.lang.String compName, java.lang.String objName, java.lang.String bopName, java.lang.String levelName, java.lang.String[ ] discSet)
static int	**changeCompartmentOwner**(java.lang.String SAName, java.lang.String compName, java.lang.String newOwnerName)
static int	**changeCompartmentOwnershipRestrictions**(java.lang.String S AName, java.lang.String compName, java.lang.String ownerName, java.lang.String[ ] ownerCompOps, java.lang.String[] ownerGiveCompOps, java.lang.String[ ] ownerSpecificCompOps)

static int	**changeUtilizersDefault**(java.lang.String ownerName, java.lang.String compName, java.lang.String utilName, java.lang.String bopName, java.lang.String levelName, java.lang.String[ ] discSet)
static int	**changeUtilizersSecurityLevel**(java.lang.String ownerName, java.lang.String compName, java.lang.String utilName, java.lang.String levelName)
static int	**createCompartment**(java.lang.String SAName, java.lang.String compName, java.lang.String ownerName, java.lang.String[ ] utilizerNames, java.lang.String[ ] levelNames, int[ ] levelDegrees, java.lang.String[ ] utilizerLevels, java.lang.String[ ] basicopNames, java.lang.String[ ] opNames, java.lang.String[ ][ ] opBops, java.lang.String[ ][ ][ ] utilizerDiscDefaults, java.lang.String[ ][ ] utilizerMandDefaults, java.lang.String[ ][ ] utilizerCompOps, java.lang.String[ ] ownerCompOps, java.lang.String[ ] ownerGiveCompOps, java.lang.String[ ] ownerSpecificCompOps, java.lang.String acmSchema)
static int	**disableActor**(java.lang.String SAName, java.lang.String actorName)
static int	**disableCompartment**(java.lang.String SAName, java.lang.String compName)
static int	**disableObject**(java.lang.String SAName, java.lang.String compName, java.lang.String objName)
static int	**enableActor**(java.lang.String SAName, java.lang.String actorName)
static int	**enableCompartment**(java.lang.String SAName, java.lang.String compName)
static int	**enableObject**(java.lang.String SAName, java.lang.String compName, java.lang.String objName)
static int	**giveUtilizersCompartmentOperationRight**(java.lang.String ownerName, java.lang.String compName, java.lang.String utilName, java.lang.String coOpName)
static int	**hasRight**(java.lang.String actorName, java.lang.String compName, java.lang.String objName, java.lang.String opName)
static int	**removeActor**(java.lang.String SAName, java.lang.String actorName)
static int	**removeCompartment**(java.lang.String SAName,

	java.lang.String compName)
static int	**removeFromBlacklist**(java.lang.String SAName, java.lang.String bopName, java.lang.String objName, java.lang.String compName, java.lang.String actorName)
static int	**removeObject**(java.lang.String ownerName, java.lang.String compName, java.lang.String objName)
static int	**removeOperation**(java.lang.String SAName, java.lang.String compName, java.lang.String opName)
static int	**removeSubject**(java.lang.String SAName, java.lang.String subjectName)
static int	**removeUtilizerActor**(java.lang.String ownerName, java.lang.String compName, java.lang.String utilName)

Figure 4.5.1: API Methods

CHAPTER 5

CASE STUDIES

5.1. Aim of Case Studies

The power of the model should be expressed by real life examples and this chapter is written to express full functionality of the model. Three case studies should be examined in comparison to existing access control models.

First case study reveals the power of blacklist structure. In the example, blacklist entry guards all future access to the object. University department is selected as example domain. In the example, criticism letter for an academician cannot be read by her, even one day she becomes the head of department.

In second case, a software company is selected as domain and as the requirement of the domain; mandatory access control is the main access control schema of the domain nevertheless the model is configured as to serve discretionary exceptions to mandatory model.

Last case is in the same domain while it reflects the personal storage of the domain. As the domain is personal, discretionary access control is the dominant schema in compartment whereas mandatory regulations limit the freedom of discretionary access control. With this feature users cannot share even their personal objects even with their friends outside the project.

5.2 Case Study Examples

5.2.1 Blacklist Example

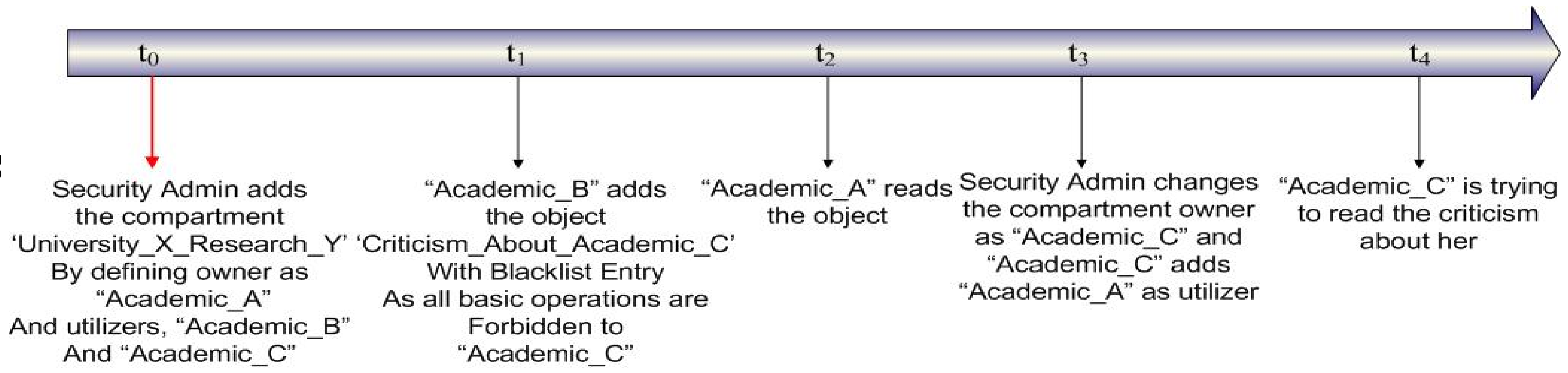

t_0
t_1
t_2
t_3
t_4
Security Admin adds the compartment 'University_X_Research_Y' By defining owner as "Academic_A" And utilizers, "Academic_B" And "Academic_C"
"Academic_B" adds the object 'Criticism_About_Academic_C' With Blacklist Entry As all basic operations are Forbidden to "Academic_C"
"Academic_A" reads the object
Security Admin changes the compartment owner as "Academic_C" and "Academic_C" adds "Academic_A" as utilizer
"Academic_C" is trying to read the criticism about her

Figure 5.1: Blacklist Example

Compartment:	**University_X_Research_Y**
Owner:	**Academic_A**
Utilizers:	**Academic_B, Academic_C**
Basic Operations:	**'read', 'write'**
Operations:	**{'read'}, {'write'}**
ACM_Schema:	**Discretionary or Mandatory**
Security_Levels:	**Owner_Specific, Top_Secret, Secret (Decreasing Privillege Order)**

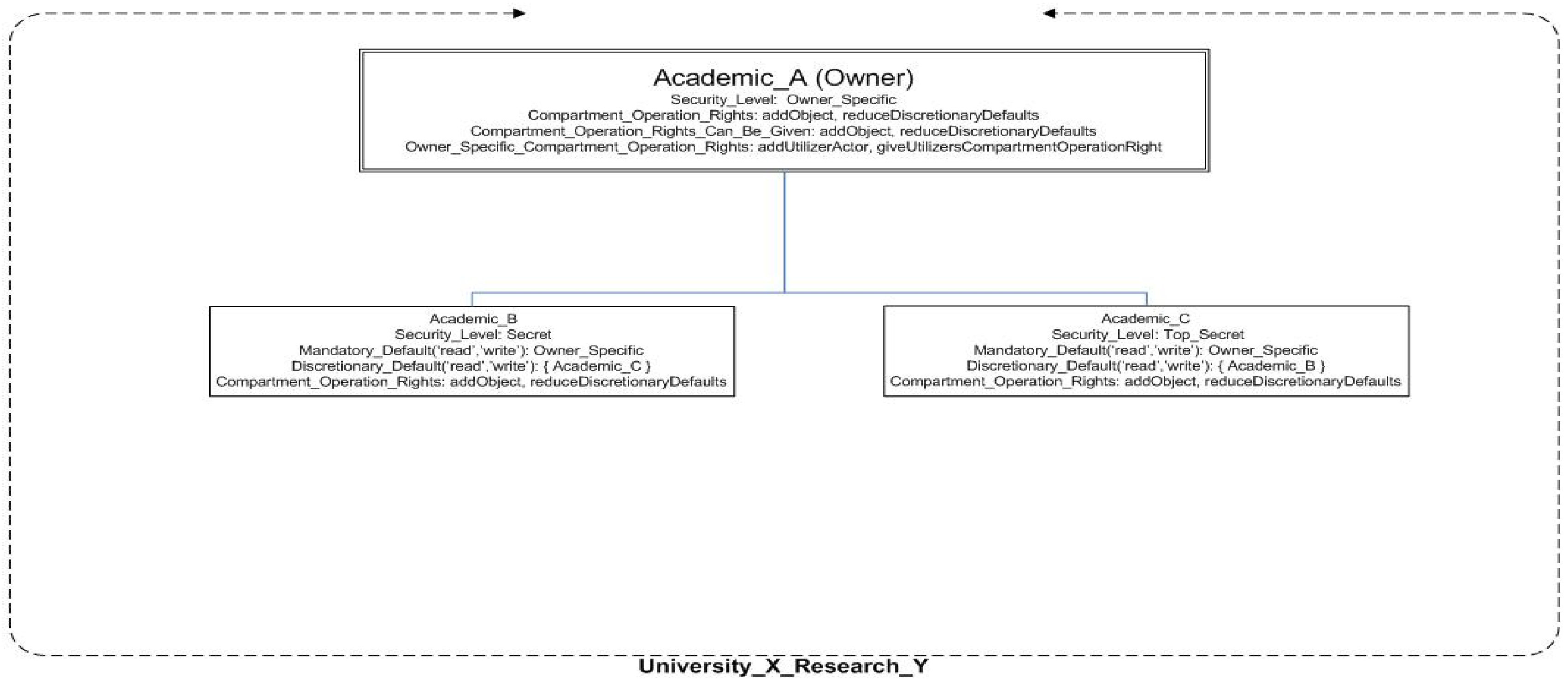

Academic_A (Owner)
Security_Level: Owner_Specific
Compartment_Operation_Rights: addObject, reduceDiscretionaryDefaults
Compartment_Operation_Rights_Can_Be_Given: addObject, reduceDiscretionaryDefaults
Owner_Specific_Compartment_Operation_Rights: addUtilizerActor, giveUtilizersCompartmentOperationRight
Academic_B
Security_Level: Secret
Mandatory_Default('read','write'): Owner_Specific
Discretionary_Default('read','write'): { Academic_C }
Compartment_Operation_Rights: addObject, reduceDiscretionaryDefaults
Academic_C
Security_Level: Top_Secret
Mandatory_Default('read','write'): Owner_Specific
Discretionary_Default('read','write'): { Academic_B }
Compartment_Operation_Rights: addObject, reduceDiscretionaryDefaults
University_X_Research_Y

Figure 5.2: Blacklist Example (cont'd)

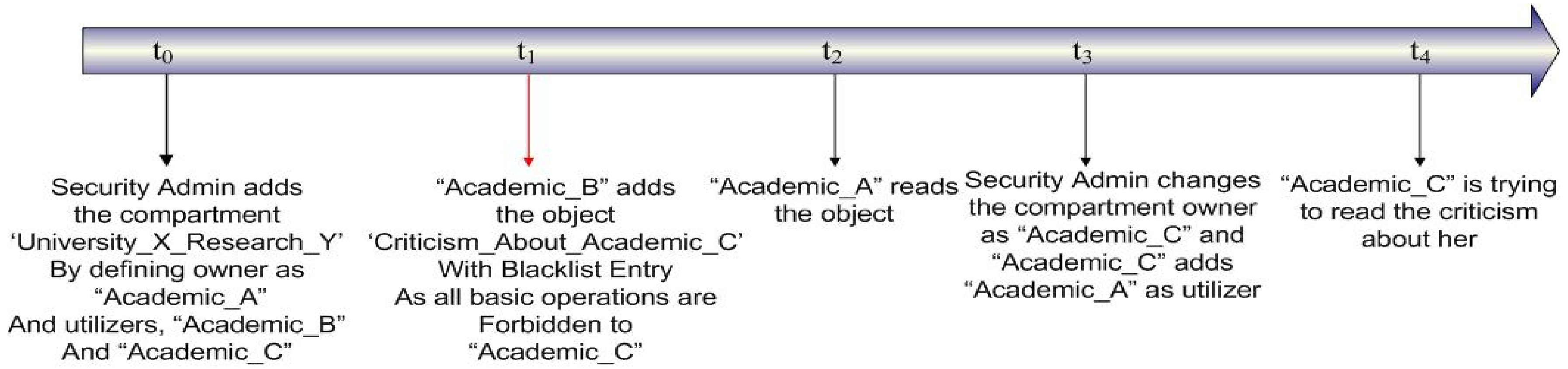
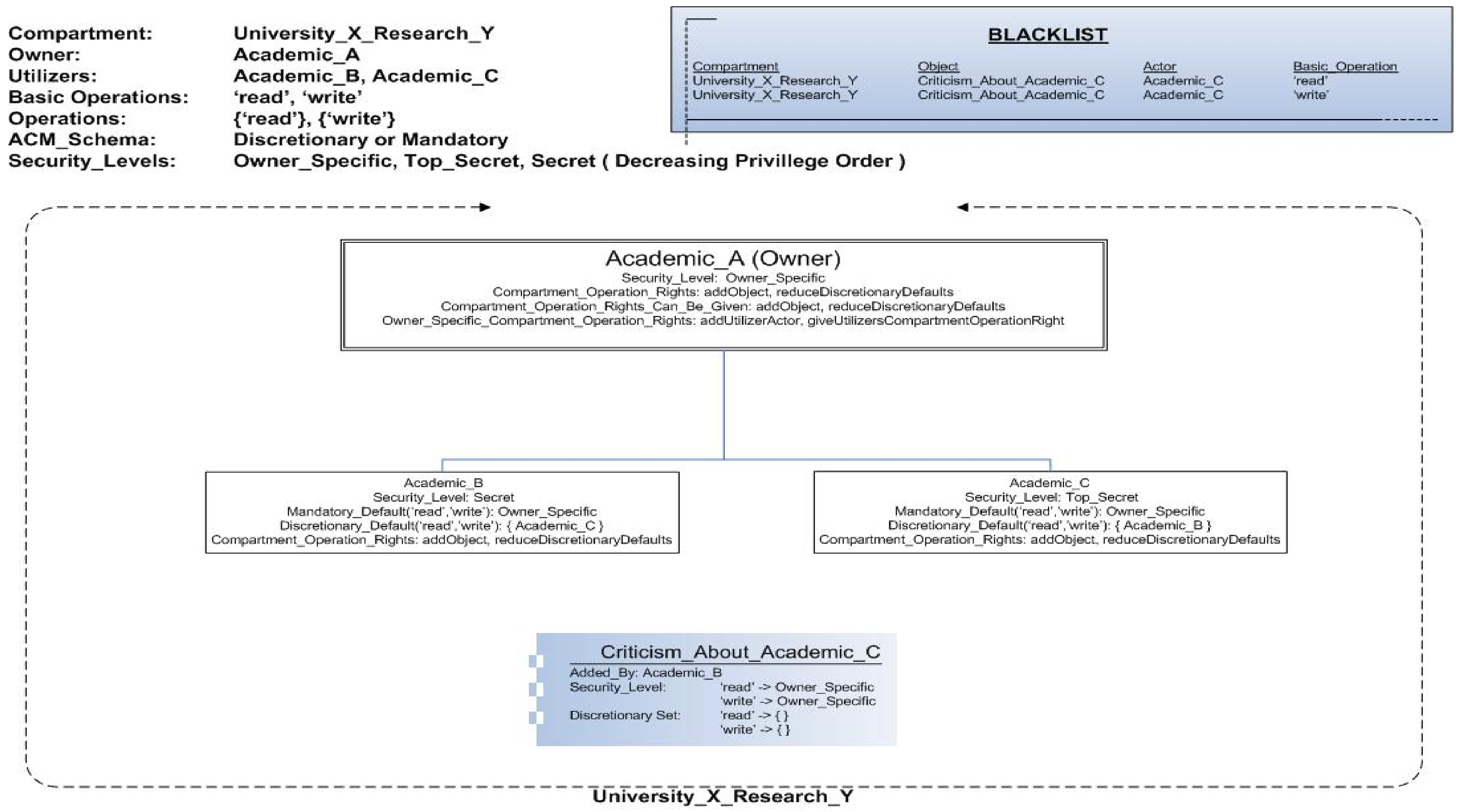

t0
t1
t2
t3
t4
Security Admin adds the compartment 'University_X_Research_Y' By defining owner as "Academic_A" And utilizers, "Academic_B" And "Academic_C"
"Academic_B" adds the object 'Criticism_About_Academic_C' With Blacklist Entry As all basic operations are Forbidden to "Academic_C"
"Academic_A" reads the object
Security Admin changes the compartment owner as "Academic_C" and "Academic_C" adds "Academic_A" as utilizer
"Academic_C" is trying to read the criticism about her

Figure 5.3: Blacklist Example (cont'd)



Compartment:	**University_X_Research_Y**
Owner:	**Academic_A**
Utilizers:	**Academic_B, Academic_C**
Basic Operations:	**'read', 'write'**
Operations:	**{'read'}, {'write'}**
ACM_Schema:	**Discretionary or Mandatory**
Security_Levels:	**Owner_Specific, Top_Secret, Secret (Decreasing Privillege Order)**

BLACKLIST

Compartment	Object	Actor	Basic_Operation
University_X_Research_Y	Criticism_About_Academic_C	Academic_C	'read'
University_X_Research_Y	Criticism_About_Academic_C	Academic_C	'write'

Academic_A (Owner)
Security_Level: Owner_Specific
Compartment_Operation_Rights: addObject, reduceDiscretionaryDefaults
Compartment_Operation_Rights_Can_Be_Given: addObject, reduceDiscretionaryDefaults
Owner_Specific_Compartment_Operation_Rights: addUtilizerActor, giveUtilizersCompartmentOperationRight
Academic_B
Security_Level: Secret
Mandatory_Default('read','write'): Owner_Specific
Discretionary_Default('read','write'): { Academic_C }
Compartment_Operation_Rights: addObject, reduceDiscretionaryDefaults
Academic_C
Security_Level: Top_Secret
Mandatory_Default('read','write'): Owner_Specific
Discretionary_Default('read','write'): { Academic_B }
Compartment_Operation_Rights: addObject, reduceDiscretionaryDefaults
Criticism_About_Academic_C
Added_By: Academic_B
Security_Level: 'read' -> Owner_Specific
'write' -> Owner_Specific
Discretionary Set: 'read' -> { }
'write' -> { }
University_X_Research_Y

Figure 5.4: Blacklist Example (cont'd)

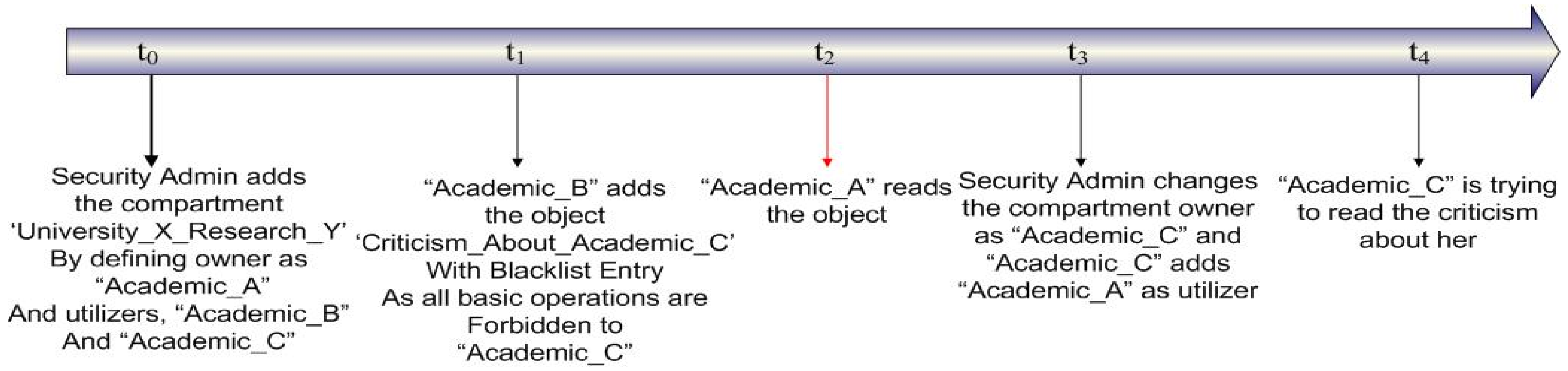
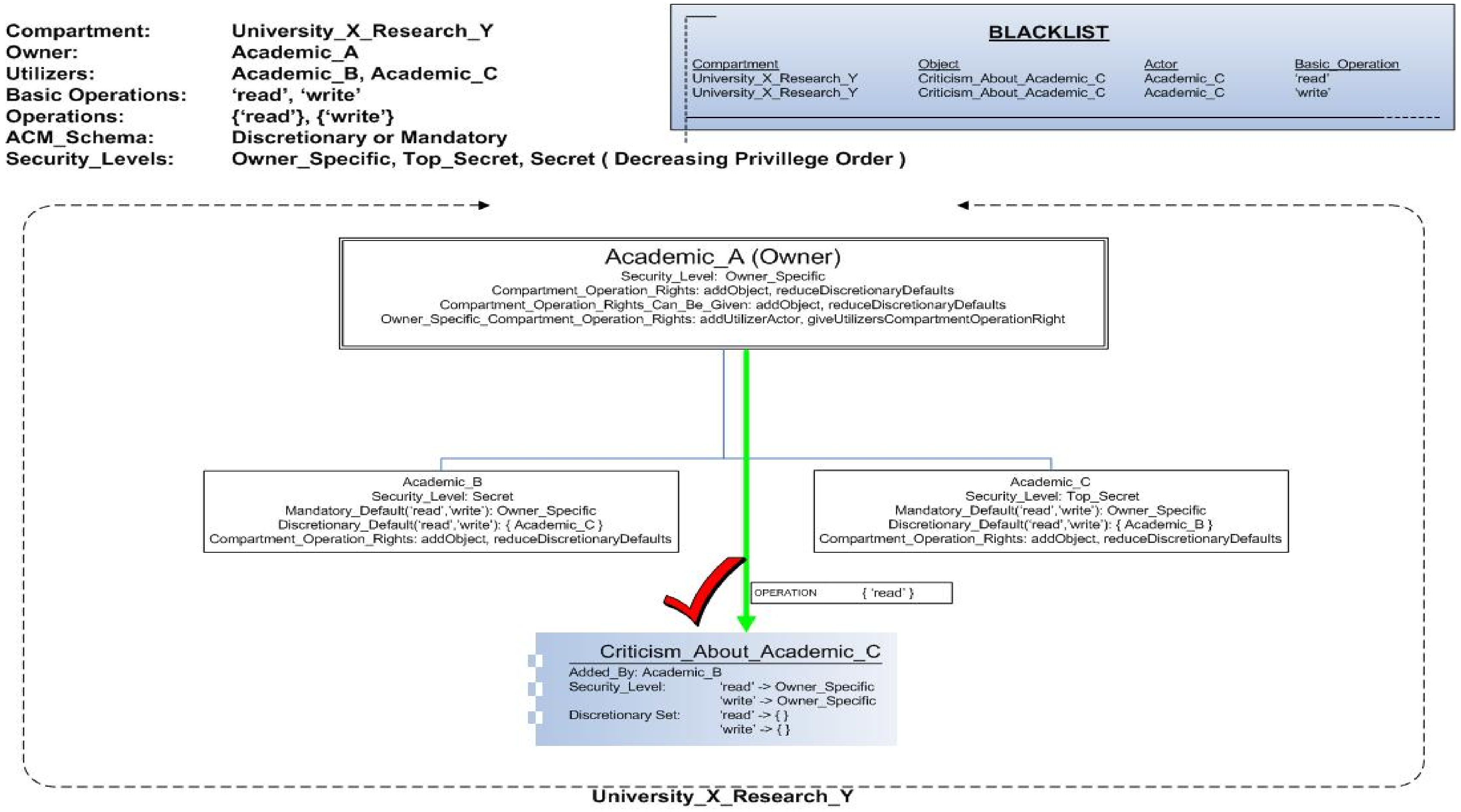

t0
t1
t2
t3
t4
Security Admin adds the compartment 'University_X_Research_Y' By defining owner as "Academic_A" And utilizers, "Academic_B" And "Academic_C"
"Academic_B" adds the object 'Criticism_About_Academic_C' With Blacklist Entry As all basic operations are Forbidden to "Academic_C"
"Academic_A" reads the object
Security Admin changes the compartment owner as "Academic_C" and "Academic_C" adds "Academic_A" as utilizer
"Academic_C" is trying to read the criticism about her

Figure 5.5: Blacklist Example (cont'd)



Compartment:	**University_X_Research_Y**
Owner:	**Academic_A**
Utilizers:	**Academic_B, Academic_C**
Basic Operations:	**‘read’, ‘write’**
Operations:	**{‘read’}, {‘write’}**
ACM_Schema:	**Discretionary or Mandatory**
Security_Levels:	**Owner_Specific, Top_Secret, Secret (Decreasing Privillege Order)**

BLACKLIST

Compartment	Object	Actor	Basic_Operation
University_X_Research_Y	Criticism_About_Academic_C	Academic_C	‘read’
University_X_Research_Y	Criticism_About_Academic_C	Academic_C	‘write’

Academic_A (Owner)
Security_Level: Owner_Specific
Compartment_Operation_Rights: addObject, reduceDiscretionaryDefaults
Compartment_Operation_Rights_Can_Be_Given: addObject, reduceDiscretionaryDefaults
Owner_Specific_Compartment_Operation_Rights: addUtilizerActor, giveUtilizersCompartmentOperationRight
Academic_B
Security_Level: Secret
Mandatory_Default(‘read’,‘write’): Owner_Specific
Discretionary_Default(‘read’,‘write’): { Academic_C }
Compartment_Operation_Rights: addObject, reduceDiscretionaryDefaults
Academic_C
Security_Level: Top_Secret
Mandatory_Default(‘read’,‘write’): Owner_Specific
Discretionary_Default(‘read’,‘write’): { Academic_B }
Compartment_Operation_Rights: addObject, reduceDiscretionaryDefaults
OPERATION { ‘read’ }
Criticism_About_Academic_C
Added_By: Academic_B
Security_Level: ‘read’ -> Owner_Specific
‘write’ -> Owner_Specific
Discretionary Set: ‘read’ -> { }
‘write’ -> { }
University_X_Research_Y

Figure 5.6: Blacklist Example (cont’d)

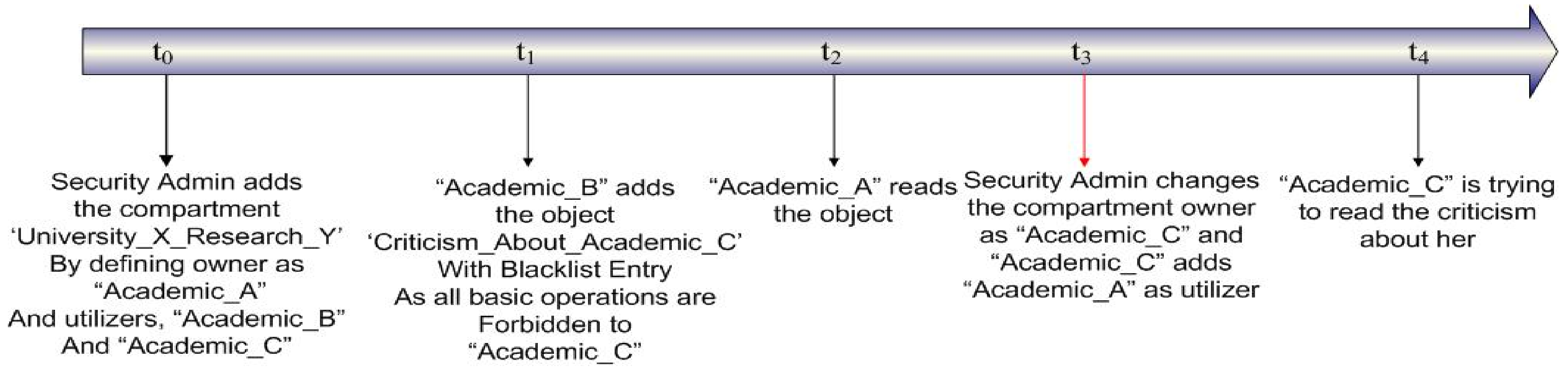
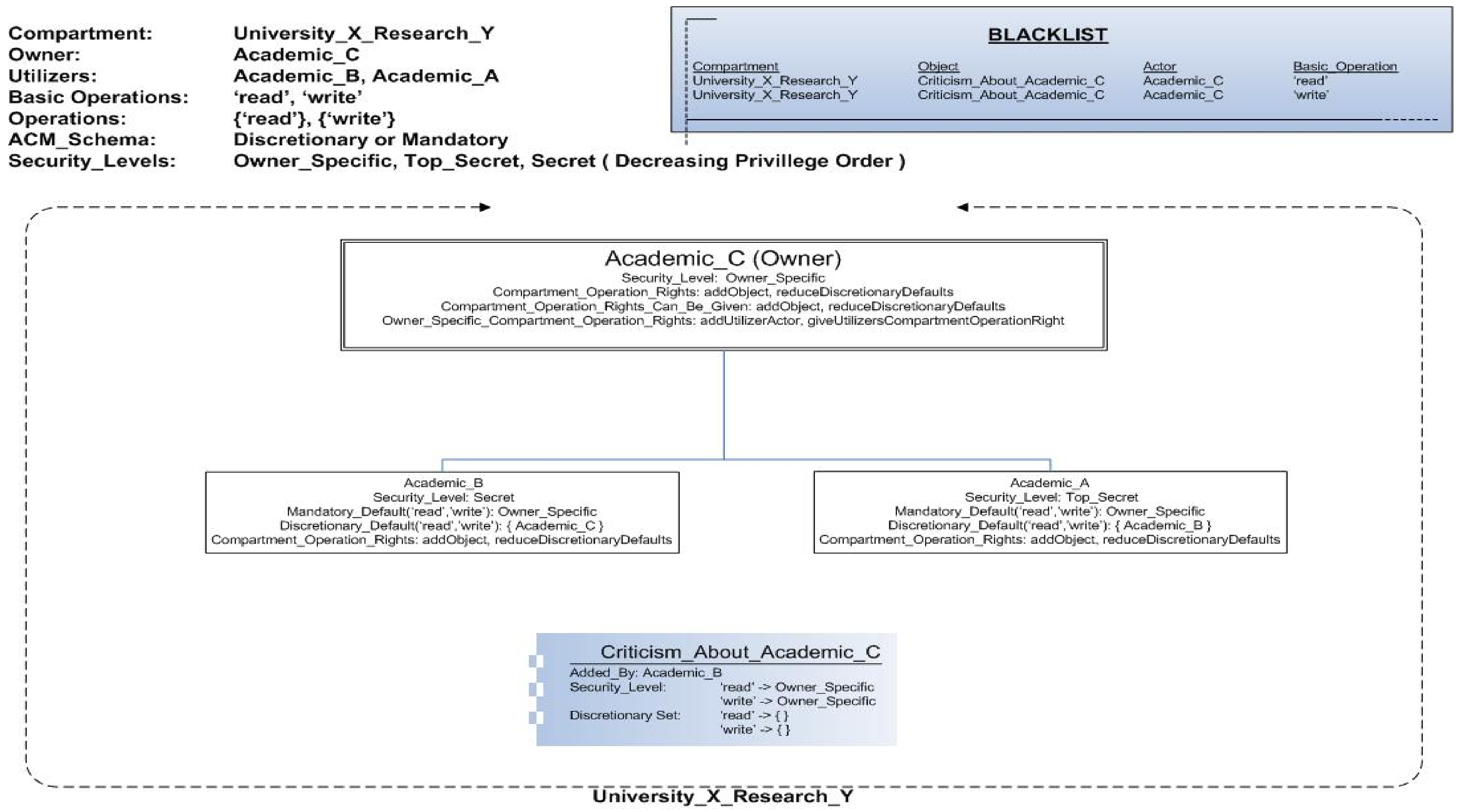

t_0
t_1
t_2
t_3
t_4
Security Admin adds the compartment 'University_X_Research_Y' By defining owner as "Academic_A" And utilizers, "Academic_B" And "Academic_C"
"Academic_B" adds the object 'Criticism_About_Academic_C' With Blacklist Entry As all basic operations are Forbidden to "Academic_C"
"Academic_A" reads the object
Security Admin changes the compartment owner as "Academic_C" and "Academic_C" adds "Academic_A" as utilizer
"Academic_C" is trying to read the criticism about her

Figure 5.7: Blacklist Example (cont'd)


Compartment: University_X_Research_Y
Owner: Academic_C
Utilizers: Academic_B, Academic_A
Basic Operations: 'read', 'write'
Operations: {'read'}, {'write'}
ACM_Schema: Discretionary or Mandatory
Security_Levels: Owner_Specific, Top_Secret, Secret (Decreasing Privillege Order)
BLACKLIST
Academic_C (Owner)
Security_Level: Owner_Specific
Compartment_Operation_Rights: addObject, reduceDiscretionaryDefaults
Compartment_Operation_Rights_Can_Be_Given: addObject, reduceDiscretionaryDefaults
Owner_Specific_Compartment_Operation_Rights: addUtilizerActor, giveUtilizersCompartmentOperationRight
Academic_B
Security_Level: Secret
Mandatory_Default('read','write'): Owner_Specific
Discretionary_Default('read','write'): { Academic_C }
Compartment_Operation_Rights: addObject, reduceDiscretionaryDefaults
Academic_A
Security_Level: Top_Secret
Mandatory_Default('read','write'): Owner_Specific
Discretionary_Default('read','write'): { Academic_B }
Compartment_Operation_Rights: addObject, reduceDiscretionaryDefaults
Criticism_About_Academic_C
Added_By: Academic_B
Security_Level: 'read' -> Owner_Specific
'write' -> Owner_Specific
Discretionary Set: 'read' -> { }
'write' -> { }
University_X_Research_Y

Compartment	Object	Actor	Basic_Operation
University_X_Research_Y	Criticism_About_Academic_C	Academic_C	'read'
University_X_Research_Y	Criticism_About_Academic_C	Academic_C	'write'

Figure 5.8: Blacklist Example (cont'd)

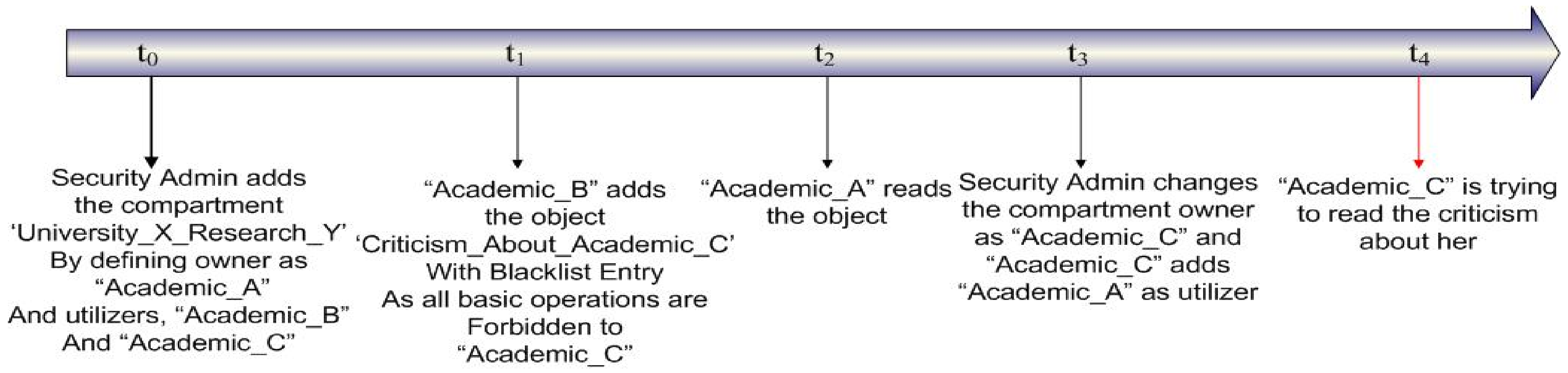
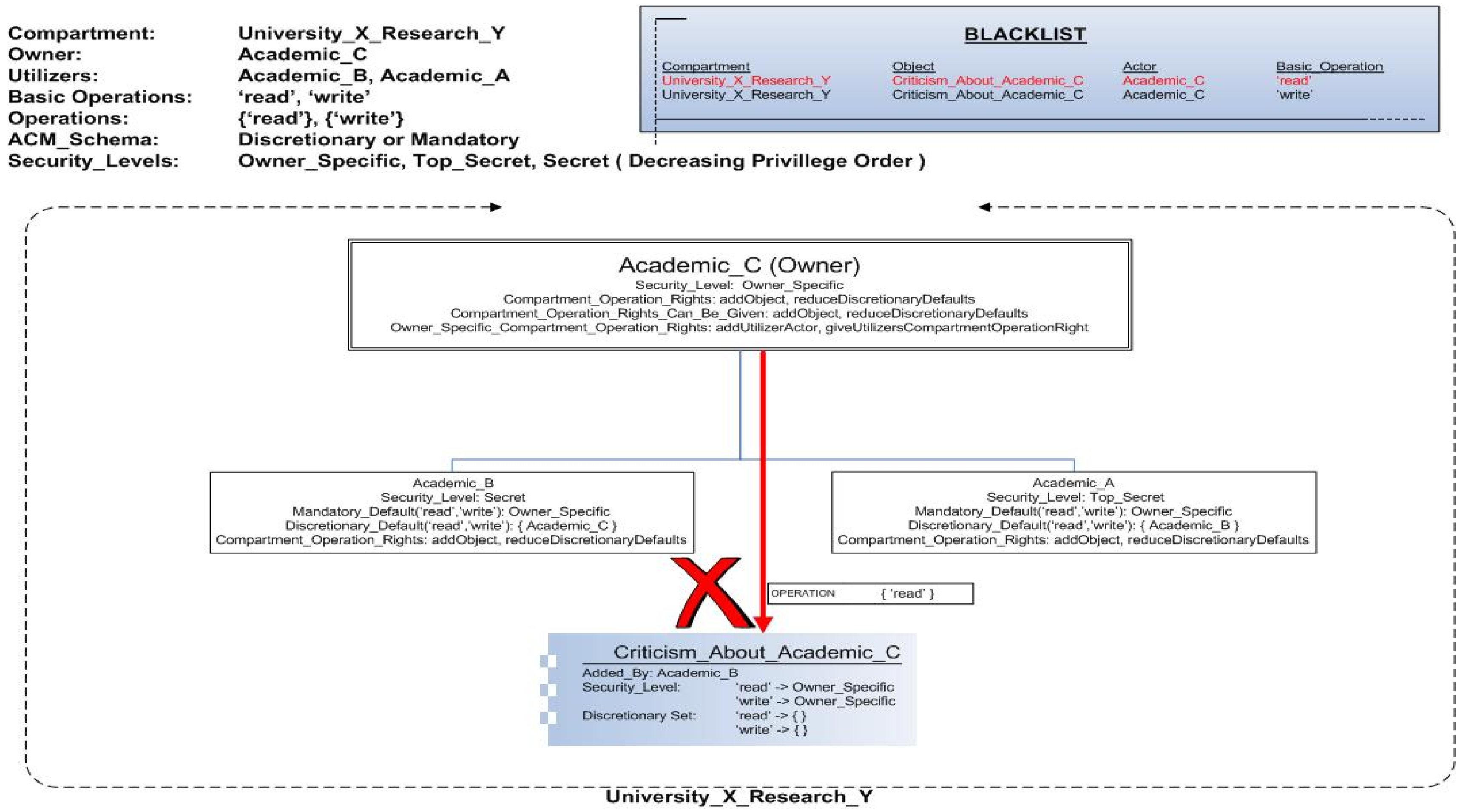

t0
t1
t2
t3
t4
Security Admin adds the compartment 'University_X_Research_Y' By defining owner as "Academic_A" And utilizers, "Academic_B" And "Academic_C"
"Academic_B" adds the object 'Criticism_About_Academic_C' With Blacklist Entry As all basic operations are Forbidden to "Academic_C"
"Academic_A" reads the object
Security Admin changes the compartment owner as "Academic_C" and "Academic_C" adds "Academic_A" as utilizer
"Academic_C" is trying to read the criticism about her

Figure 5.9: Blacklist Example (cont'd)



Compartment:	**University_X_Research_Y**
Owner:	**Academic_C**
Utilizers:	**Academic_B, Academic_A**
Basic Operations:	**‘read’, ‘write’**
Operations:	**{‘read’}, {‘write’}**
ACM_Schema:	**Discretionary or Mandatory**
Security_Levels:	**Owner_Specific, Top_Secret, Secret (Decreasing Privillege Order)**

BLACKLIST

Compartment	Object	Actor	Basic_Operation
University_X_Research_Y	Criticism_About_Academic_C	Academic_C	'read'
University_X_Research_Y	Criticism_About_Academic_C	Academic_C	'write'

Academic_C (Owner)
Security_Level: Owner_Specific
Compartment_Operation_Rights: addObject, reduceDiscretionaryDefaults
Compartment_Operation_Rights_Can_Be_Given: addObject, reduceDiscretionaryDefaults
Owner_Specific_Compartment_Operation_Rights: addUtilizerActor, giveUtilizersCompartmentOperationRight
Academic_B
Security_Level: Secret
Mandatory_Default('read','write'): Owner_Specific
Discretionary_Default('read','write'): { Academic_C }
Compartment_Operation_Rights: addObject, reduceDiscretionaryDefaults
Academic_A
Security_Level: Top_Secret
Mandatory_Default('read','write'): Owner_Specific
Discretionary_Default('read','write'): { Academic_B }
Compartment_Operation_Rights: addObject, reduceDiscretionaryDefaults
X
OPERATION { 'read' }
Criticism_About_Academic_C
Added_By: Academic_B
Security_Level: 'read' -> Owner_Specific
'write' -> Owner_Specific
Discretionary Set: 'read' -> { }
'write' -> { }
University_X_Research_Y

Figure 5.10: Blacklist Example (cont’d)

5.2.2 MAC Exception Example

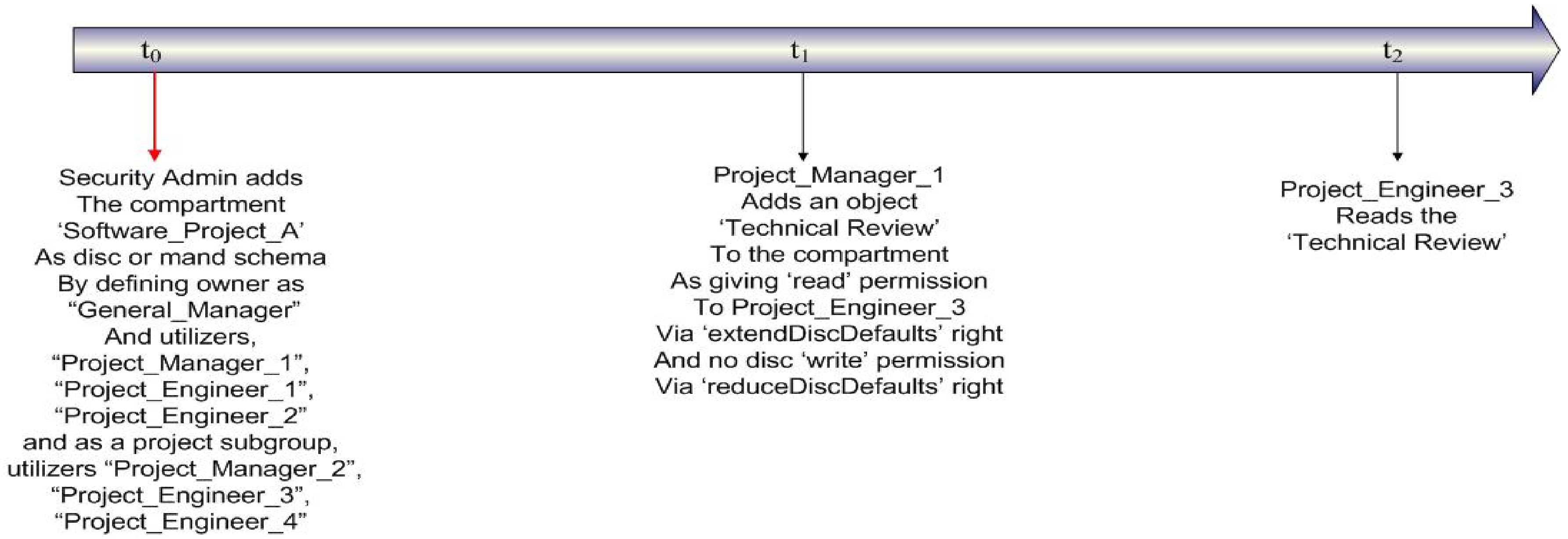

t_0
t_1
t_2
Security Admin adds The compartment 'Software_Project_A' As disc or mand schema By defining owner as "General_Manager" And utilizers, "Project_Manager_1", "Project_Engineer_1", "Project_Engineer_2" and as a project subgroup, utilizers "Project_Manager_2", "Project_Engineer_3", "Project_Engineer_4"
Project_Manager_1 Adds an object 'Technical Review' To the compartment As giving 'read' permission To Project_Engineer_3 Via 'extendDiscDefaults' right And no disc 'write' permission Via 'reduceDiscDefaults' right
Project_Engineer_3 Reads the 'Technical Review'

Figure 5.11: MAC Exception Example

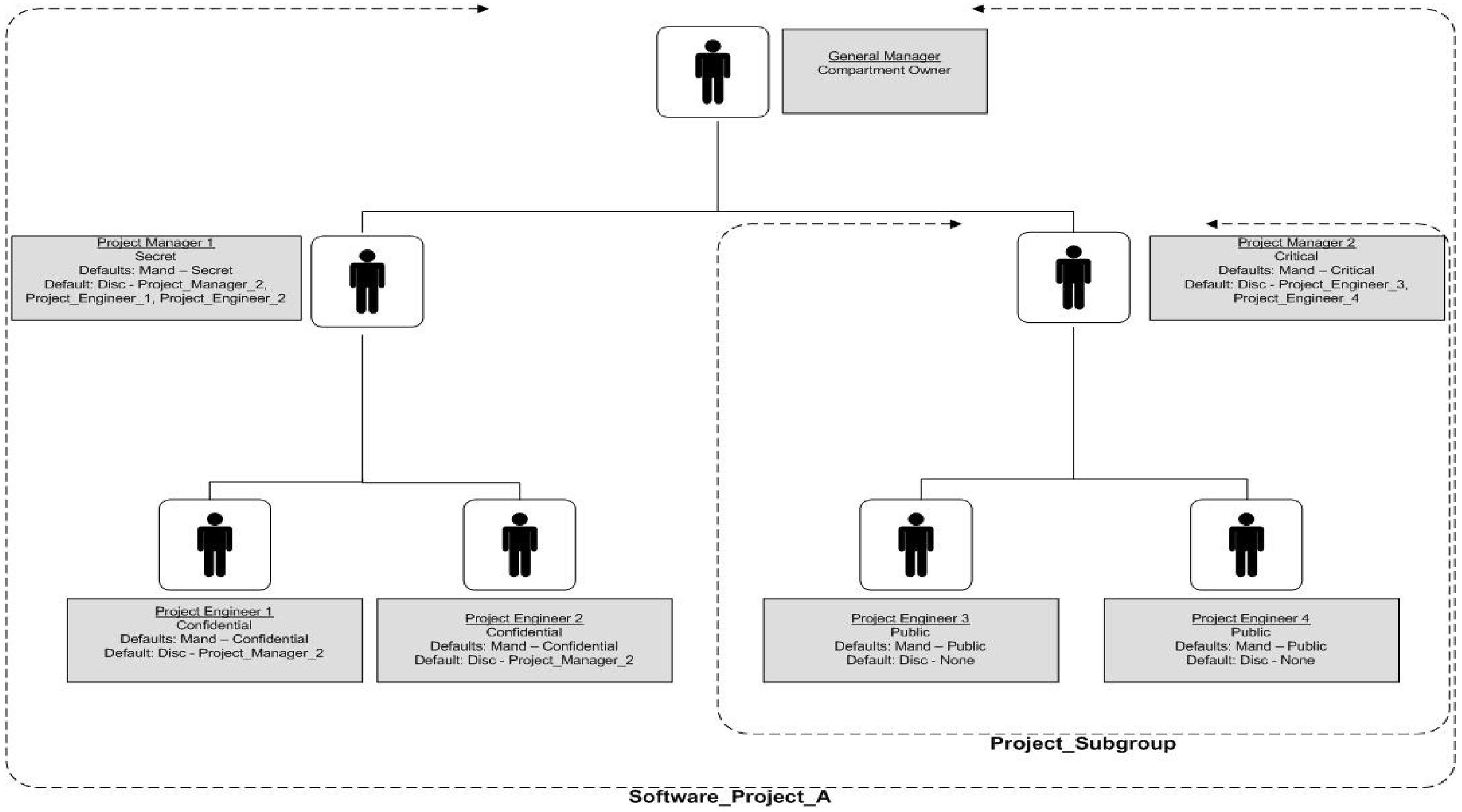


Figure 5.12: MAC Exception Example (cont'd)

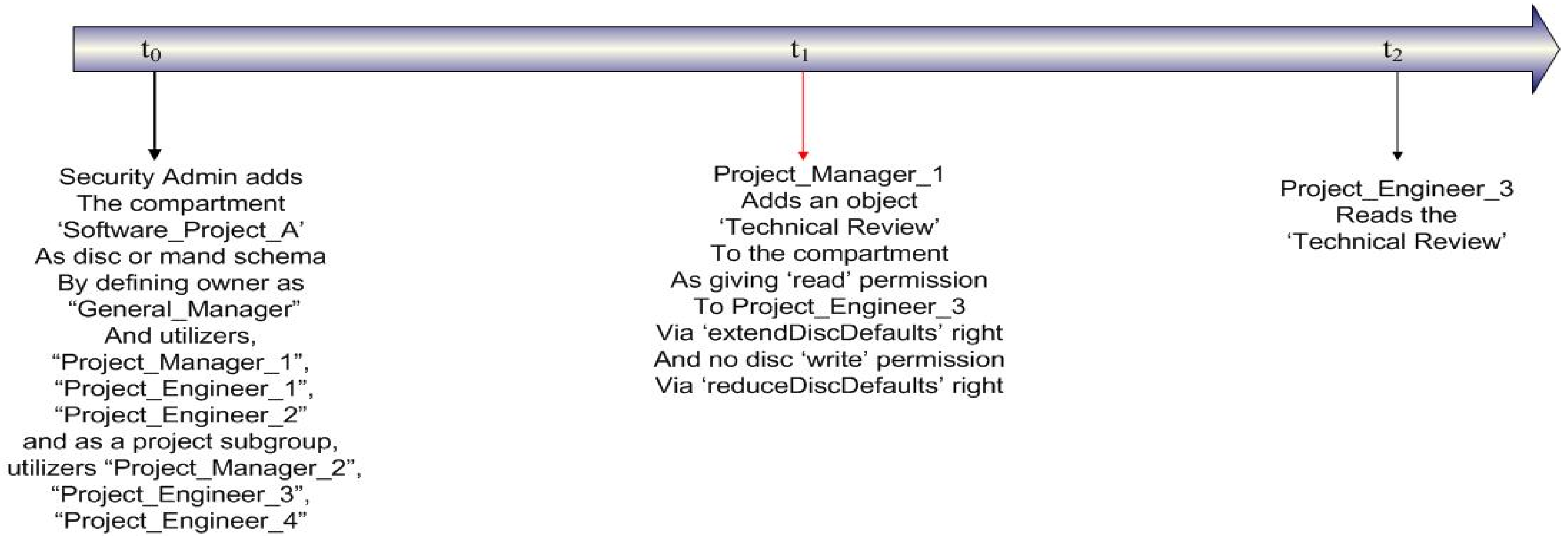


Figure 5.13: MAC Exception Example (cont'd)

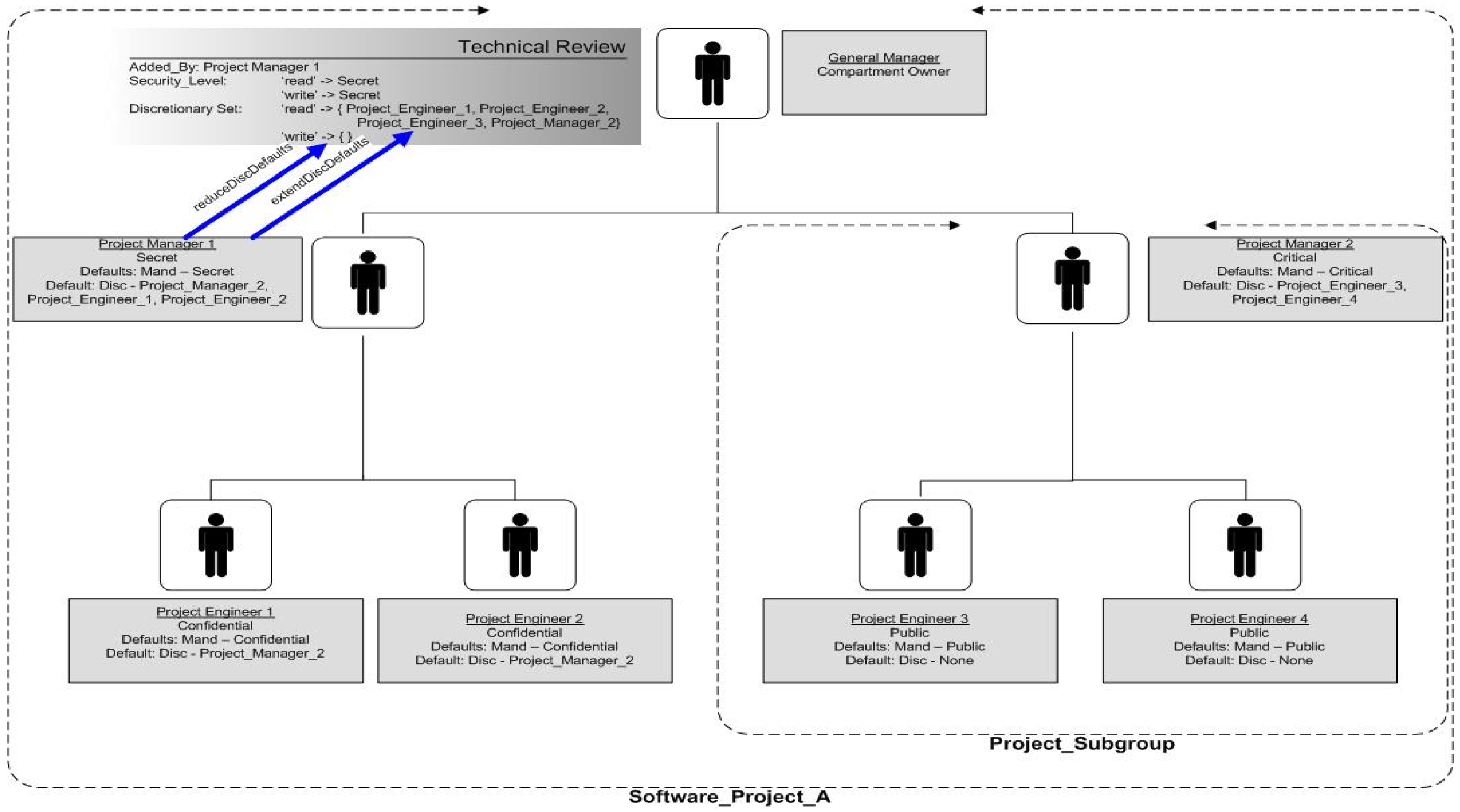

Technical Review
Added_By: Project Manager 1
Security_Level: 'read' -> Secret
'write' -> Secret
Discretionary Set: 'read' -> { Project_Engineer_1, Project_Engineer_2, Project_Engineer_3, Project_Manager_2}
'write' -> { }
reduceDiscDefaults
extendDiscDefaults
General Manager
Compartment Owner
Project Manager 1
Secret
Defaults: Mand – Secret
Default: Disc - Project_Manager_2, Project_Engineer_1, Project_Engineer_2
Project Manager 2
Critical
Defaults: Mand – Critical
Default: Disc - Project_Engineer_3, Project_Engineer_4
Project Engineer 1
Confidential
Defaults: Mand – Confidential
Default: Disc - Project_Manager_2
Project Engineer 2
Confidential
Defaults: Mand – Confidential
Default: Disc - Project_Manager_2
Project Engineer 3
Public
Defaults: Mand – Public
Default: Disc - None
Project Engineer 4
Public
Defaults: Mand – Public
Default: Disc - None
Project_Subgroup
Software_Project_A

Figure 5.14: MAC Exception Example (cont'd)

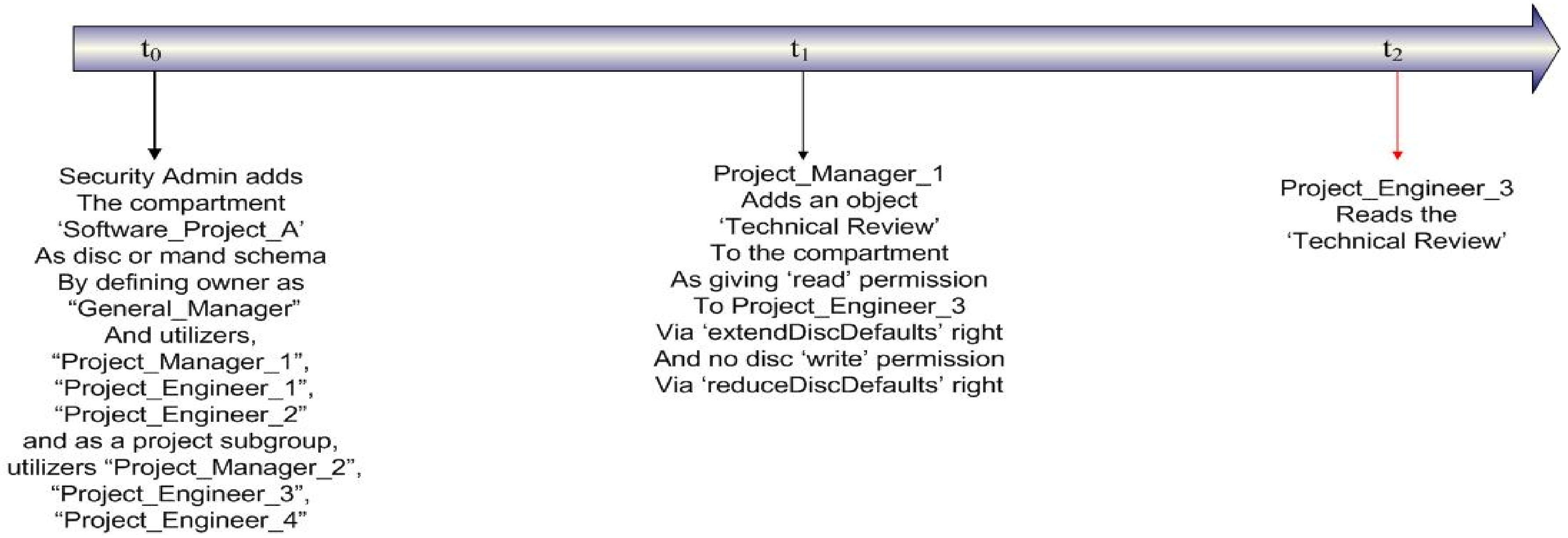


Figure 5.15: MAC Exception Example (cont'd)

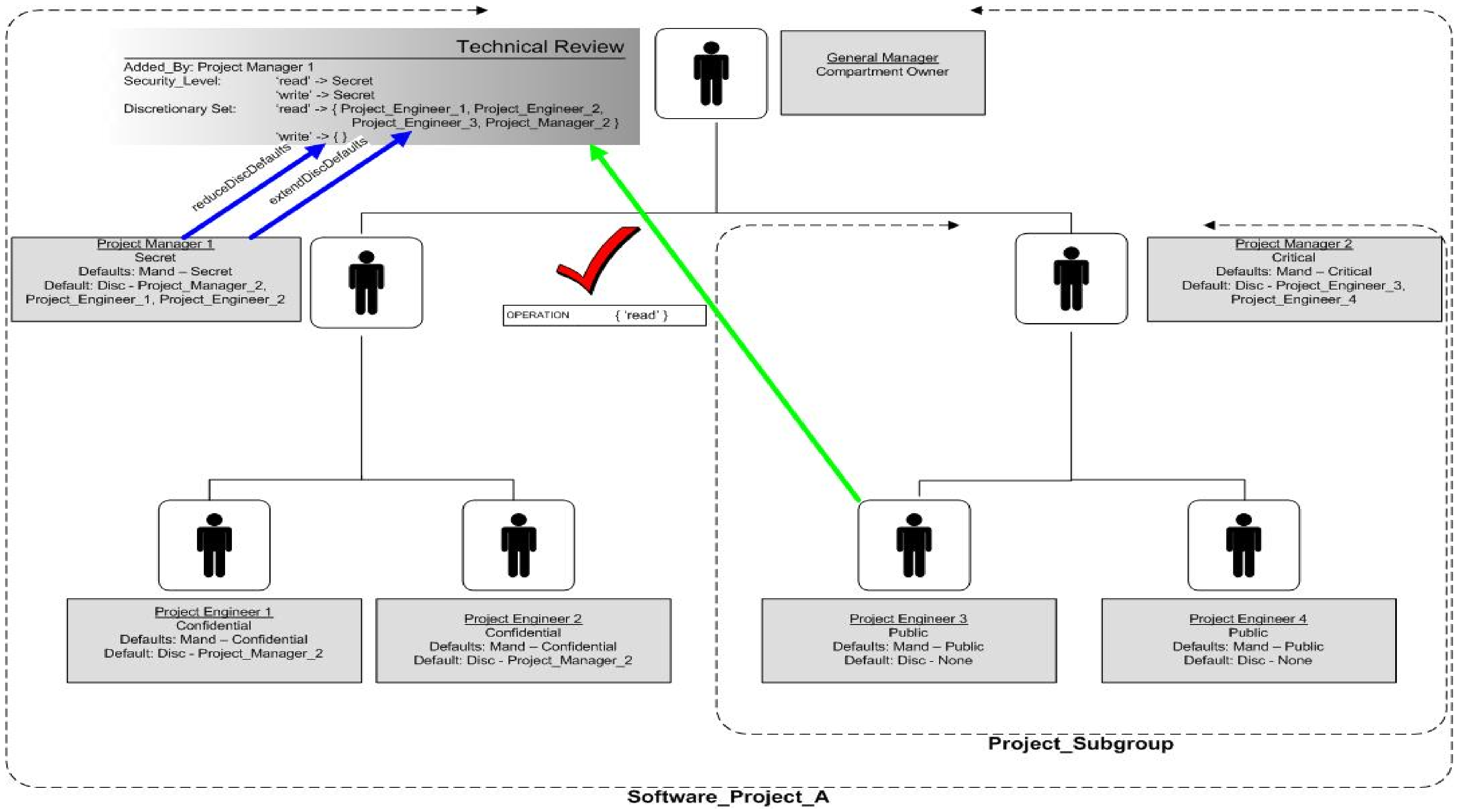

Technical Review
Added_By: Project Manager 1
Security_Level: 'read' -> Secret
'write' -> Secret
Discretionary Set: 'read' -> { Project_Engineer_1, Project_Engineer_2, Project_Engineer_3, Project_Manager_2 }
'write' -> { }
reduceDiscDefaults
extendDiscDefaults
General Manager
Compartment Owner
Project Manager 1
Secret
Defaults: Mand – Secret
Default: Disc - Project_Manager_2, Project_Engineer_1, Project_Engineer_2
OPERATION { 'read' }
Project Manager 2
Critical
Defaults: Mand – Critical
Default: Disc - Project_Engineer_3, Project_Engineer_4
Project Engineer 1
Confidential
Defaults: Mand – Confidential
Default: Disc - Project_Manager_2
Project Engineer 2
Confidential
Defaults: Mand – Confidential
Default: Disc - Project_Manager_2
Project Engineer 3
Public
Defaults: Mand – Public
Default: Disc - None
Project Engineer 4
Public
Defaults: Mand – Public
Default: Disc - None
Project_Subgroup
Software_Project_A

Figure 5.16: MAC Exception Example (cont'd)

5.2.3 DAC Restriction Example

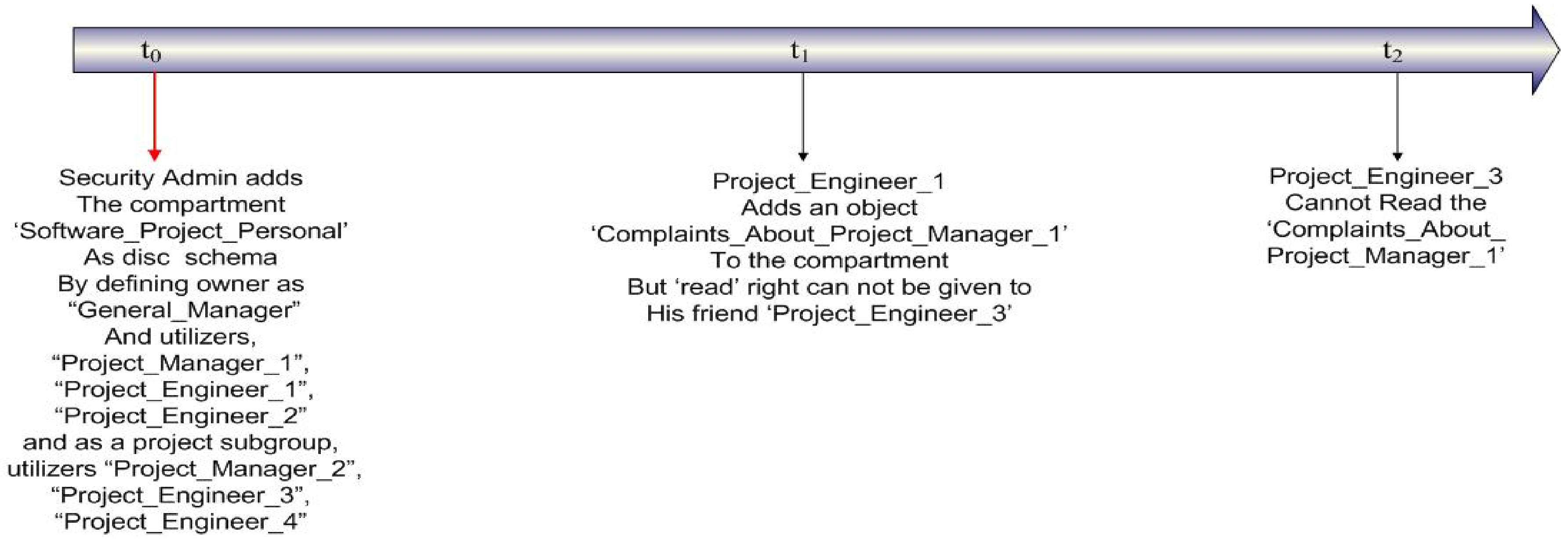


Figure 5.17: DAC Restriction Example

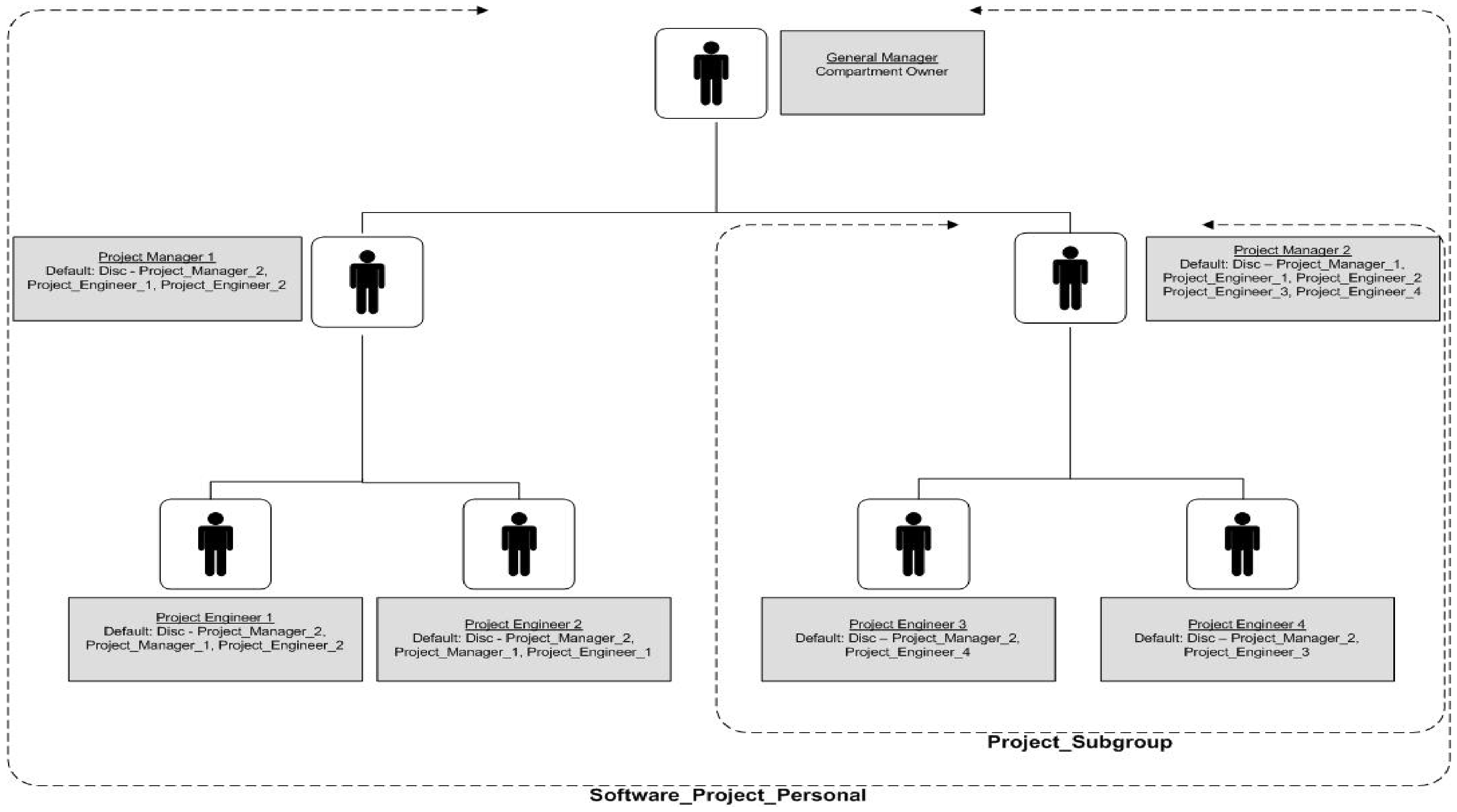

General Manager
Compartment Owner
Project Manager 1
Default: Disc - Project_Manager_2, Project_Engineer_1, Project_Engineer_2
Project Manager 2
Default: Disc – Project_Manager_1, Project_Engineer_1, Project_Engineer_2 Project_Engineer_3, Project_Engineer_4
Project Engineer 1
Default: Disc - Project_Manager_2, Project_Manager_1, Project_Engineer_2
Project Engineer 2
Default: Disc - Project_Manager_2, Project_Manager_1, Project_Engineer_1
Project Engineer 3
Default: Disc – Project_Manager_2, Project_Engineer_4
Project Engineer 4
Default: Disc – Project_Manager_2, Project_Engineer_3
Project_Subgroup
Software_Project_Personal

Figure 5.18: DAC Restriction Example (cont'd)

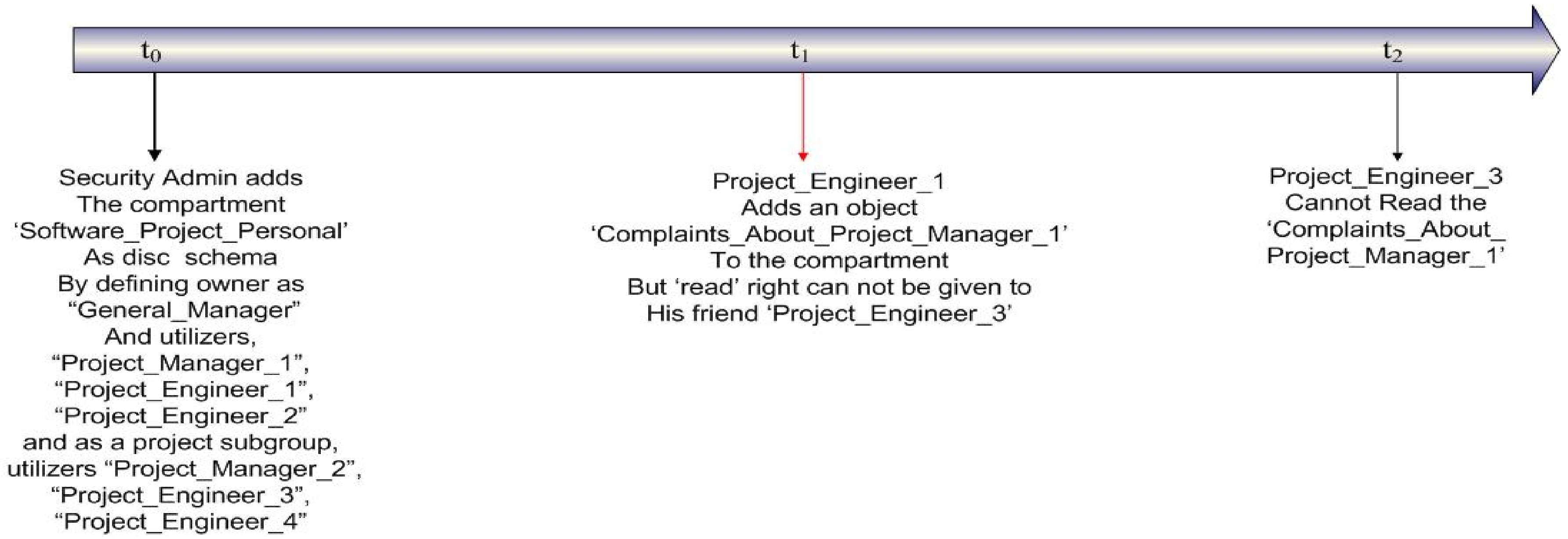

t0
t1
t2
Security Admin adds The compartment 'Software_Project_Personal' As disc schema By defining owner as "General_Manager" And utilizers, "Project_Manager_1", "Project_Engineer_1", "Project_Engineer_2" and as a project subgroup, utilizers "Project_Manager_2", "Project_Engineer_3", "Project_Engineer_4"
Project_Engineer_1 Adds an object 'Complaints_About_Project_Manager_1' To the compartment But 'read' right can not be given to His friend 'Project_Engineer_3'
Project_Engineer_3 Cannot Read the 'Complaints_About_Project_Manager_1'

Figure 5.19: DAC Restriction Example (cont'd)

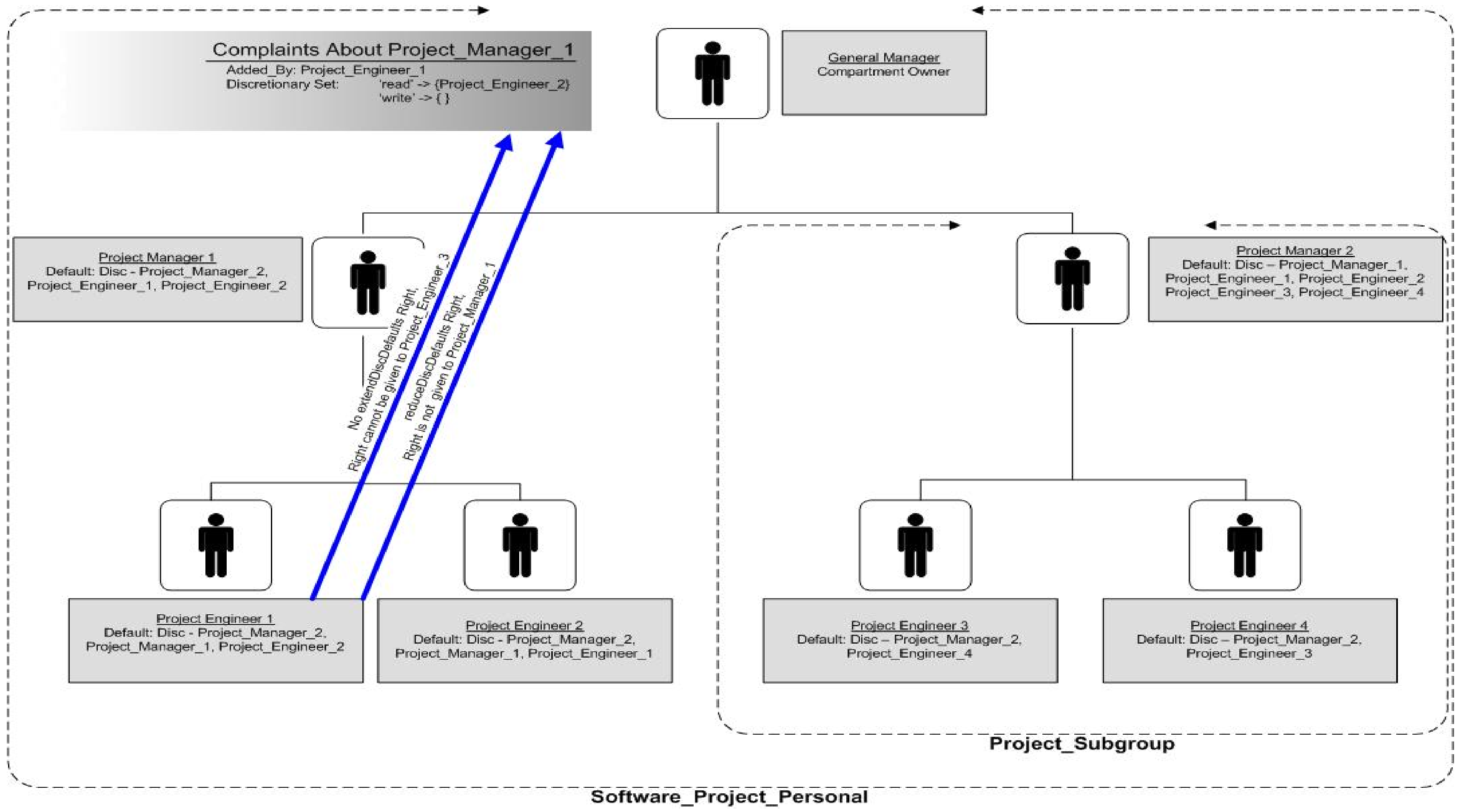

Complaints About Project_Manager_1
Added_By: Project_Engineer_1
Discretionary Set: 'read' -> {Project_Engineer_2}
'write' -> { }
General Manager
Compartment Owner
Project Manager 1
Default: Disc - Project_Manager_2, Project_Engineer_1, Project_Engineer_2
No extendDiscDefaults Right, Right cannot be given to Project_Engineer_3
reduceDiscDefaults Right, Right is not given to Project_Manager_1
Project Manager 2
Default: Disc – Project_Manager_1, Project_Engineer_1, Project_Engineer_2 Project_Engineer_3, Project_Engineer_4
Project Engineer 1
Default: Disc - Project_Manager_2, Project_Manager_1, Project_Engineer_2
Project Engineer 2
Default: Disc - Project_Manager_2, Project_Manager_1, Project_Engineer_1
Project Engineer 3
Default: Disc – Project_Manager_2, Project_Engineer_4
Project Engineer 4
Default: Disc – Project_Manager_2, Project_Engineer_3
Project_Subgroup
Software_Project_Personal

Figure 5.20: DAC Restriction Example (cont'd)

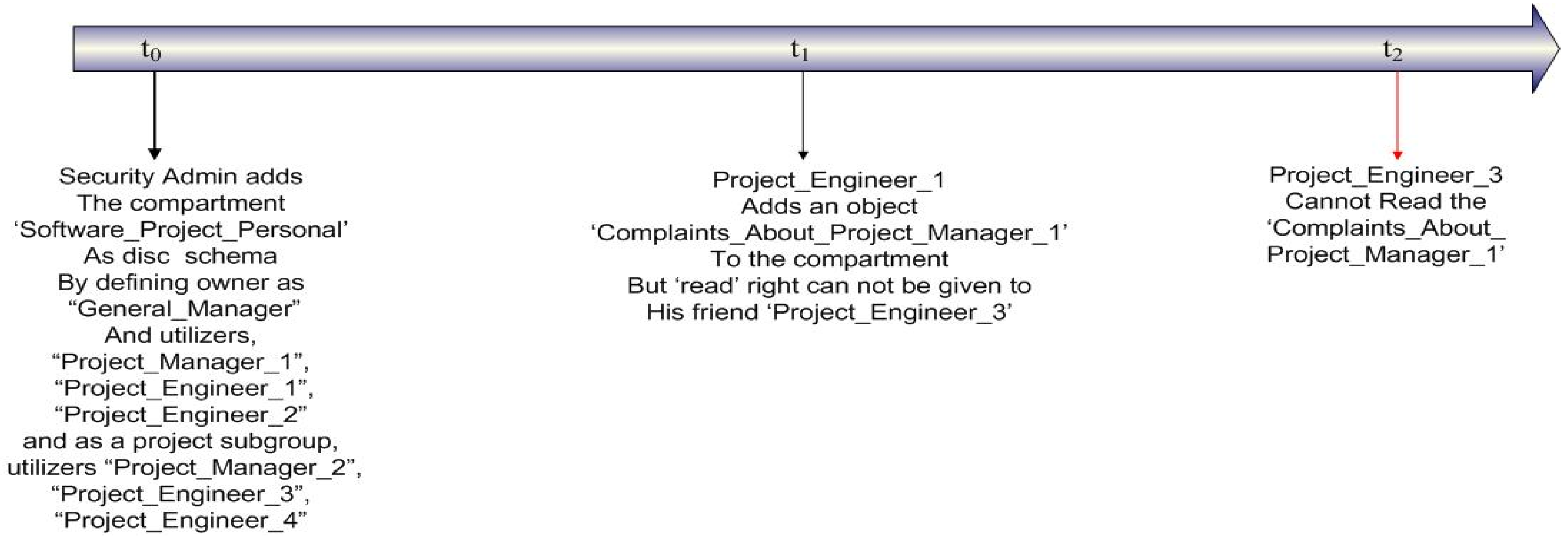

t$_0$
t$_1$
t$_2$
Security Admin adds The compartment 'Software_Project_Personal' As disc schema By defining owner as "General_Manager" And utilizers, "Project_Manager_1", "Project_Engineer_1", "Project_Engineer_2" and as a project subgroup, utilizers "Project_Manager_2", "Project_Engineer_3", "Project_Engineer_4"
Project_Engineer_1 Adds an object 'Complaints_About_Project_Manager_1' To the compartment But 'read' right can not be given to His friend 'Project_Engineer_3'
Project_Engineer_3 Cannot Read the 'Complaints_About_Project_Manager_1'

Figure 5.21: DAC Restriction Example (cont'd)

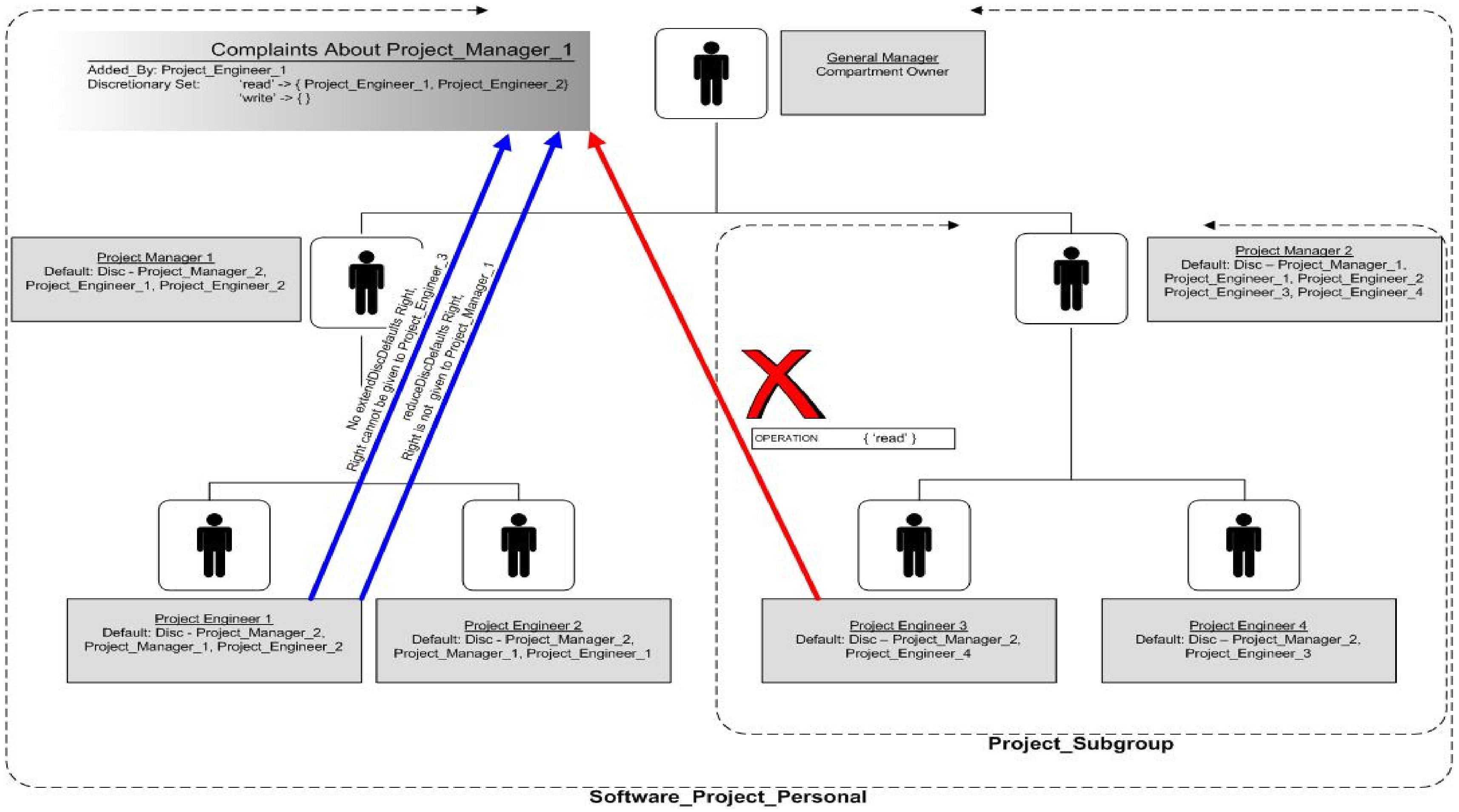

Complaints About Project_Manager_1
Added_By: Project_Engineer_1
Discretionary Set: 'read' -> { Project_Engineer_1, Project_Engineer_2}
'write' -> { }
General Manager
Compartment Owner
Project Manager 1
Default: Disc - Project_Manager_2, Project_Engineer_1, Project_Engineer_2
Project Manager 2
Default: Disc – Project_Manager_1, Project_Engineer_1, Project_Engineer_2 Project_Engineer_3, Project_Engineer_4
No extendDiscDefaults Right, Right cannot be given to Project_Engineer_3
reduceDiscDefaults Right, Right is not given to Project_Manager_1
OPERATION { 'read' }
Project Engineer 1
Default: Disc - Project_Manager_2, Project_Manager_1, Project_Engineer_2
Project Engineer 2
Default: Disc - Project_Manager_2, Project_Manager_1, Project_Engineer_1
Project Engineer 3
Default: Disc – Project_Manager_2, Project_Engineer_4
Project Engineer 4
Default: Disc – Project_Manager_2, Project_Engineer_3
Project_Subgroup
Software_Project_Personal

Figure 5.22: DAC Restriction Example (cont'd)

CHAPTER 6

SUMMARY, CONCLUSIONS & FUTURE WORK

6.1. Summary

To sum up; after a brief literature survey on access control mechanisms and their combination, a new configurable and hybrid access control model has been built with its formal background. The model is primarily focused on the defined requirements in real life and three major case studies have been investigated in order to express the power of the model in the process of handling real life requirements. Implementation details of the model have been given and efficiency considerations have been made.

6.2. Conclusions

The need for a hybrid access control model has been increasing with the increasing demand on complex organizations containing multiple sub-organizations with different duties. A hybrid access control model for this need has been defined, mathematical and implementation details behind the model have been described in this work. The model is aimed to handle the requirements of the systems that contain independent modules and a hybrid access control approach containing discretionary and mandatory in each model. The model is designed to handle exceptions for each access control schema. Verification of the model has partly been done by case studies nevertheless a complete verification of the model is to be discussed in a different study because of its complexity.

The implementation details of the model can be changed according to the environment and the target usage. The system can be used in every platform whose access operations to sources should be controlled by a high level system. The main advantage of using the system is to enrich mandatory regulations by adding discretionary exceptions and to bound discretionary freedom with mandatory regulations. In addition, other concepts such as blacklists and compartment operations access control have been defined and integrated to the model. The model can be used as an external server to query access control requests and modifications where the integrated usage as an API has already been supported. The model builds a modular base for this approach; many concepts and approaches are going to be added to the model to fulfill the requirements of different domains as a future work.

6.3. Future Work

The system has been designed as a base unit for further development and the modular architecture of the components makes improvement easier and fast. The overall view of the model to the access control domain resides on combining mandatory and discretionary approaches on a single and configurable basis. Therefore other approaches on different dimensions of the model can be extended and improved in a straightforward manner. The main targets of the future work of the model should be seen from this perspective.

First of all, in many systems there is not a clear distinction between objects and subjects. The role meanings are exactly different nevertheless the roles can interchange for an action for a specific entry. The model resolves this issue by the requirement of defining the entity as an object and a subject independently. The dynamic type property can be added to the system as determining security rules and roles dynamically according to action performed [26]. In addition to that role based access control can be added to the system by defining a layer between subjects and roles for arranging all security rules and settings according to those. Temporal issues and periodicity constraints can be added to the model; as role assignment, blacklist entries, permissions can be made temporal [5, 6]. Provisional approaches can be made concrete as actions have prerequisite actions to be permitted [15].

Finally, the term “compartment” can be enriched within an organizational point of view [1]. In the model, compartments are independent entries where all security rules and schemas are defined for each compartment differently. The term organization includes a hierarchical structure including compartment-to-compartment relationships. This improvement makes the model much more modular nevertheless the efficiency and time complexity should be clearly investigated before.

REFERENCES

[1] A. Abou El Kalam, R. El Baida, P. Balbiani, S. Benferhat, F. Cuppens, Y. Deswarte, A. Mi`ege, C. Saurel, and G. Trouessin. Organization Based Access Control. In Proceedings of IEEE 4th International Workshop on Policies for Distributed Systems and Networks, Lake Como, Italy, June 2003.

[2] D.E., Bell, and L.J. LaPadula. Secure computer systems: mathematical foundations and model. M74-244, The MITRE Corp., Bedford, Mass., May 1973.

[3] E. Bertino, B. Catania, E. Ferrari, and P. Perlasca. A System to Specify and Manage Multipolicy Access Control Models. In Proceedings of the IEEE 3rd International Workshop on Policies for Distributed Systems and Networks, 2002.

[4] E. Bertino, P. Samarati, S. D. C. D. Vimercati, and E. Ferrari. Exception based information flow control in object-oriented systems. ACM Transactions on Information and System Security (TISSEC), 1998.

[5] E. Bertino, C. Bettini, E. Ferrari, and P. Samarati. An access control model supporting periodicity constraints and temporal reasoning. ACM Trans. Database Syst. 23, 3, 231–285, 1998.

[6] E. Bertino, P. Bonatti, and E. Ferrari. TRBAC: A temporal role-based access control model. In Proceedings of the Fifth ACM Workshop on Role Based Access Control, 21–30, 2000.

[7] K. J. Biba. Integrity considerations for secure computer systems. Technical Report TR-3153, The Mitre Corporation, Bedford, MA, April 1977.

[8] W. E. Boebert, and C. T. Ferguson. A partial solution to the discretionary Trojan horse problem. In Proc. of the 8th Nat. Computer Security Conf., pages 141–144, Gaithersburg, MD, 1985.

[9] D. Brewer and M. Nash. The Chinese wall security policy. In Proceedings of the Symposium on Security and Privacy, IEEE Press, Los Alamitos, Calif., pp. 215–228., 1989.

[10] E. Cohen, R. K. Thomas, W. Winsborough, and D. Shands. Models for coalition-based access control (CBAC). In Proceedings of the seventh ACM Symposium on Access Control models and technologies (SACMAT), pp. 97–105, June 2002.

[11] G. S. Graham, and P. J.Denning. Protection—principles and practice. AFIPS Conf. Proc., Vol. 40, SJCC, AFIPS Press, Montvale, N.J., pp. 417--429., 1972.

[12] M. A. Harrison, W. L. Ruzzo, and J.D. Ullman. On protection in operating systems. Proc. Fifth Symposium on Operating Systems Principles, The University of Texas at Austin, pp. 14-24., 1975.

[13] T. Jaeger, X. Zhang, and F. Cacheda. Policy management using access control spaces. ACM Transactions on Information and System Security, 6(3):327–364, 2003.

[14] P. A. Karger. Limiting the damage potential of discretionary Trojan Horses. In Proc. IEEE Symposium on Security and Privacy, pp. 32–37, Oakland, CA, 1987.

[15] M. Kudo. Provision Based Access Control Model. International Journal of Information Security. Springer. 2002.

[16] B. Lampson. Protection. ACM Oper. Syst. Rev. 8, 1, 18–24., 1974.

[17] C. Mccollum, J. Messing, and L. Notargiacomo. Beyond the pale of MAC and DAC—Defining new forms of access control. In Proceedings of the Symposium on Security and Privacy, IEEE Press, Los Alamitos, Calif., 190–900., 1990.

[18] S. Osborn, R. Sandhu, and Q. Munawer. Configuring role-based access control to enforce mandatory and discretionary access control policies. ACM Trans. Inf. Syst. Sec. 3, 2., 2000.

[19] E. Rissanen, B. Sadighi Firozabadi, M.J. Sergot. Towards a mechanism for discretionary overriding of access control (position paper). In: Proc. 12th International Workshop on Security Protocols, Cambridge, April 2004.

[20] P. Samarati, and S. D. C. D. Vimercati. Access Control: Policies, Models, and Mechanisms. In Proc. of the 2nd FOSAD, LNCS, pages 137–196, Springer – Verlag , 2001.

[21] S. Sandhu. Lattice-based access control models. IEEE Computer, November 1993.

[22] R. Sandhu. The Typed Access Matrix Model, IEEE Symposium on Security and Privacy, 1992.

[23] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-based access control: A multi-dimensional view. In Proceedings of the 10thConference on Computer Security Applications (Dec.). IEEE Computer Society Press, Los Alamitos, CA, 54–62., 1994.

[24] R. Sandhu, and P. Samarati. Access controls, principles and practice. IEEE Communications, 32(9), pp. 40-48, 1994.

[25] A. Stoughton. Access flow: A protection model which integrates access control and information flow. In Proc. of the IEEE Symposium on Security and Privacy, pp. 9–18, Oakland, CA, 1981.

[26] J. Tidswell, and J. Potter. A Dynamically Typed Access Control Model. Proceedings of the Third Australasian Conference on information Security and Privacy, London, 1998.

[27] K. Walter, G. W. F. Ogden, W. C. Rounds, F. T. Bradshaw, S. R. Ames, and D. G. Sumaway. Primitive models for computer security. Technical Report TR ESD-TR-4-117, Case Western Reserve University, 1974.