2011

AN OBLIGATION MODEL FOR USAGE CONTROL

Jian Zhu

AN OBLIGATION MODEL FOR USAGE CONTROL

(Thesis format: Monograph)

by

Jian Zhu

Graduate Program in Computer Science

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science

The School of Graduate and Postdoctoral Studies
The University of Western Ontario
London, Ontario, Canada

THE UNIVERSITY OF WESTERN ONTARIO

School of Graduate and Postdoctoral Studies

CERTIFICATE OF EXAMINATION

Supervisor:

.....

Dr. Sylvia L. Osborn

Examiners:

.....

Dr. Michael Bauer

.....

Dr. John Barron

.....

Dr. Stuart Rankin

The thesis by

Jian Zhu

entitled:

An Obligation Model for Usage Control

is accepted in partial fulfillment of the

requirements for the degree of

Masters of Science

.....

Date

.....

Chair of the Thesis Examination Board

Abstract

How to control the access and usage of digital resources is one of the most important issues in computer security nowadays. Among them, how to control the resources when they have been passed to the client-side is a research hot spot. The Usage Control Model (UCON) has been proposed to solve this problem. In this research, we focus on one core component of the UCON model, the obligation. We propose a new obligation model to solve the problems the current ones can not deal with, especially for post-obligation. We also offer two testing scenarios, propose an architecture for a prototype based on the proposed model and apply the scenarios to the prototype architecture for proof-of-concept.

Keywords: Usage Control Model, UCON, Obligation Model

Acknowledgements

First and foremost, I want to thank my family in China for their constantly love and support. They are always there for me whenever I need them, and this thesis would never have been possible without them.

I am also so grateful to my supervisor in Western, Professor Sylvia L. Osborn, for her guidance, suggestions and always being available whenever I had problems. Because of her endless support, my journey at Western became much more easier. Her attitude to academism influences me a lot and I believe that will be the the fortune of my life.

I want to thank Hessam Zakerzadeh and Aiman Al Harbi, as the senior members when I just joined the group, they provided lots of useful information to me. They made the beginning days of my study here much easier.

Last but not least, I want to thank all my friends here in Canada, we spent a lot of time together. We encouraged each when feeling sad or lonely, without you guys, my life for the past year would be much harder.

Contents

List of Figures

List of Appendices

Chapter 1

Introduction

1.1 Motivation

How to control the access and usage of digital resources is one of the most important issues in computer security. In the past decades, different models for access control have been widely studied. Access control is a system which enables an authority to control access to areas and resources in a given physical facility or computer-based information system. In computer security, access control includes authentication, authorization and audit. Some models that have been widely used for access control include discretionary access control (DAC) [1], mandatory access control (MAC) [1] and role-based access control (RBAC) [2].

All those mechanisms for access control only focus on the control of access on server-side objects. Once an access request is granted and one copy of object is downloaded to the client-side, traditional access control will have no control over it. In order to tackle this kind of problem, a new model has been proposed, the usage control (UCON) model [3]. The term usage means usage of rights on digital objects.

The UCON model has three core components and three additional components. The core components are *Subjects*, *Rights* and *Objects* while *Authorizations*, *Obligations* and *Conditions* compose the additional components. This thesis will mainly focus on the obligation part of the UCON model.

1.2 Problem Definition

A general requirement of obligations includes actions that must be performed by a subject or the system so that the fulfillment can be checked after the access. In the initial work by Sandhu and Park for the UCON model, the authors mainly focus on the core components of the model rather than the obligation part. In that paper the main drawback of the UCON model is the inability to handle actions after the resources has been used. An example is the requirement that the user must delete a file within 30 days after he/she obtained it. Then in later work, post-obligations were added to the UCON model to deal with this problem.
The purpose of a post-obligation is twofold:

1. It can be used to execute obligation actions that are related to the current usage despite the fact that it has no affect on the decision making of the current usage.

2. It can affect future usage sessions.

Although introducing the idea of post-obligation made the UCON model much more flexible, there are several aspects which were not considered:

1. They only gave a theoretical model without any real world application scenario for

demonstration purposes or any experiments to show the model could correctly solve the problem.

2. For the enforcement mechanism, the model proposed integrating the PEP (Policy Enforcement Point) into the application. That approach is not flexible. In order to use the object, the user must use the specified application. On the other hand, this could also bring security issues, since the PEP is in the application, so if an alternative application could be provided by a third party, this new application may choose simply to ignore the obligation requirements.

3. The proposed UCON model is only suitable in a closed system. Nowadays, more and more open systems are widely used. Unlike closed systems, in this kind of system the users who require resources are not pre-defined. In order to cover this new application scenario, modifications have to be made to the obligation model.

In later work, some changes have been made to the UCON model, but the problems discussed above are still not addressed. These will be the topics that this thesis focuses on.

1.3 Aim of the Thesis

The research objectives of this thesis can be stated as follows:

1. Propose an obligation model, especially a post-obligation model, to achieve the goals as follows:

   - The system should work both in open systems and closed systems.

- The system should have a more secure enforcement mechanism. For example, the PEP may be embedded in a lower level like the operating system rather than the application level.
- If the assigning of an obligation involves a human, the delegation and revocation of duty must be considered.

2. Offer several application scenarios for testing the model.
3. Add a history-based policy engine to the model. History-based policy engine has been used in traditional access control models. In this part, additional components are added to the model for storing related session's information for later evaluation.
4. Add negotiation to the obligation model. Negotiation is a technique developed to allow peers to conduct bilateral and iterative exchanges of digital credentials to bootstrap trust relationships in open systems. For now, the study of negotiation is mainly focused on trust. Even though there are already some models for negotiation, applying those models to obligation has not studied. The model could be applied to an obligation model after some modifications. By adding negotiation to obligations, the enforcement of fulfillment of obligations will be much more flexible.

1.4 Organization of Thesis

The rest of this thesis is organized as follows:

Chapter 2 is related work; in this chapter some basic definitions and notations related to the

work in the thesis are given so the readers can better understand the rest of the thesis. Also it consists of the literature review part. We first give the definition of control domain and reference monitor; the UCON model components are then introduced in detail. Next we talk about the UCON enforcement mechanisms and real world implementations of UCON. At last the delegation and revocation of obligation and negotiation are discussed.

In Chapter 3 two application scenarios are introduced: an eHealth system as the scenario for a closed system and a cloud service system as the scenario for an open system. Firstly the definition of closed and open system will be given, next the detailed application scenarios are introduced along with summarizations of the obligations in the two scenarios.

The core part of this thesis, the obligation model, is proposed in Chapter 4. Before actually proposing the new model, the very original obligation model in UCON_{ABC} and an enhanced model given later are introduced. Then a new obligation model for usage control is described in detail.

In the fifth chapter the prototype architecture of a system is presented based on the obligation model proposed in Chapter 4 and the prototype architecture is applied to the scenarios discussed in the third chapter for proof of concept.

Finally the conclusion of this thesis and possible future research directions are discussed in Chapter 7.

Chapter 2

Related Work

In this chapter, we will introduce some related work done previously by other researchers. Topics include traditional access control model, the usage control model, delegation and revocation of obligation and negotiation.

Access control mechanisms only focus on the control of access to server-side objects. But nowadays there has been a great need for the control of objects on the client-side. In order to tackle this kind of problem, a new model is proposed, the usage control model (UCON) [3].

2.1 Traditional Access Control

Access control is a system which enables an authority to control access to areas and resources in a given physical facility or computer-based information system. In computer security, access control includes authentication, authorization and audit.

2.1.1 DAC

Discretionary Access Control (DAC) is an access policy that is determined by the owner of an object. The owner of the object decides who is allowed to access it and what privileges they have on it.

In DAC, every object has an owner that controls the permissions to access it. In most DAC systems, each object's initial owner is the subject that caused it to be created. The access policy for an object is determined by its owner. The owner of one object can assign access to it to other subjects for specific resources.

2.1.2 MAC

Compared to DAC, one could say that Mandatory Access Control (MAC) is an access policy determined by the system rather than the owner. MAC is used in multilevel systems that process highly sensitive data, such as classified government or military information. A multilevel system is a single computer system that handles multiple classification levels between subjects and objects.

In a MAC system, all subjects and objects must have labels assigned to them. A subject's sensitivity label specifies its level of trust. An object's sensitivity label specifies the level of trust required for access. In order to access a given object, the subject must have a sensitivity level equal to or higher than the requested object.

2.1.3 RBAC

In recent years, more and more people started to realize that traditional discretionary and mandatory access controls (DAC and MAC respectively) are inappropriate for the information security needs of many commercial organizations. DAC is too weak for effective control of information assets while at the same time, MAC is too strict and restrictive. It is commonly used for military classified information.

Based on the discussion above, role-based access control (RBAC) has been proposed as an alternative to traditional DAC and MAC. It represents a major advancement in flexibility and detail of control from the existing standards of DAC and MAC.

RBAC has three main components namely users, roles and privileges. In RBAC, privileges are not granted to users directly but assigned to roles. Each user is associated with one or more roles in order to get the privileges. Currently there are two mainstream categories of RBAC models-The ANSI standard model [4] and the Role Graph Model [5].

Role Graph Model

In 1994, Nyanchama and Osborn proposed the role graph model for RBAC. Based on their definition, a role graph is an acyclic, directed graph in which the nodes represent the roles in a system, and the edges represent the is-junior relationship. Every role graph has a *MaxRole* and a *MinRole*. *MaxRole* represents the union of all the privileges of the roles in the role graph. It does not need to have any users authorized to it while *MinRole* represents the minimum set of privileges available to all roles. The components of the role graph model are represented in Figure 2.1.

ANSI Standard Model

Another mainstream model in RBAC is the ANSI standard model proposed by Sandhu. Figure 2.2 shows the core components of this model.

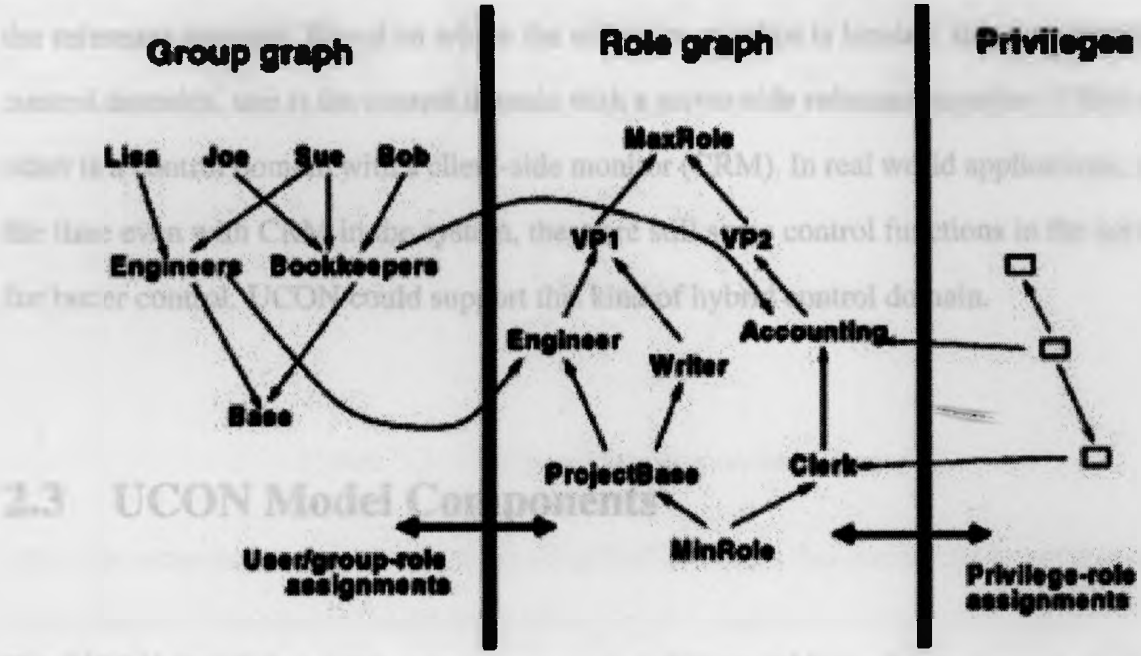

Figure 2.1: Components of the Role Graph Model (From [6])

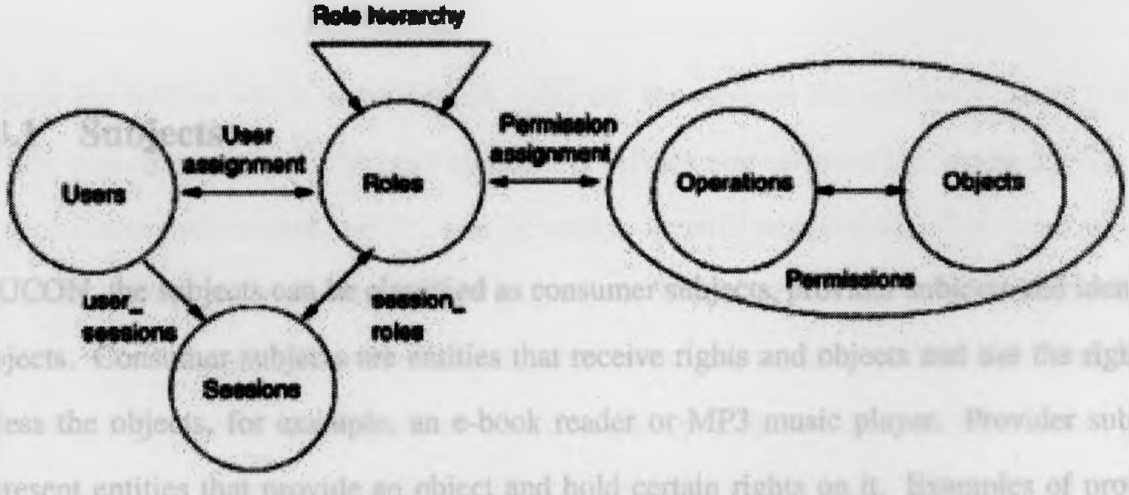

Figure 2.2: Components of ANSI RBAC (From [6])

2.2 Control Domain and Reference Monitor

Control domain [7] is an area of coverage where rights and usage of rights on digital objects are under control of a reference monitor. A reference monitor associates decision policies and rules for control of access to digital objects. Subjects can only access digital objects through the reference monitor. Based on where the reference monitor is located, there are two types of control domains, one is the control domain with a server-side reference monitor (SRM) and the other is a control domain with a client-side monitor (CRM). In real world applications, most of the time even with CRM in the system, there are still some control functions in the server side for better control. UCON could support this kind of hybrid control domain.

2.3 UCON Model Components

The UCON model has three core components and three additional components as shown in Figure 2.3. Among which the core components are subjects, rights and objects while authorizations, obligations and conditions compose the additional components.

2.3.1 Subjects

In UCON, the subjects can be classified as consumer subjects, provider subjects and identifier subjects. Consumer subjects are entities that receive rights and objects and use the rights to access the objects, for example, an e-book reader or MP3 music player. Provider subjects represent entities that provide an object and hold certain rights on it. Examples of provider subjects include an author of an e-book, a distributor of the book, a primary care physician. The

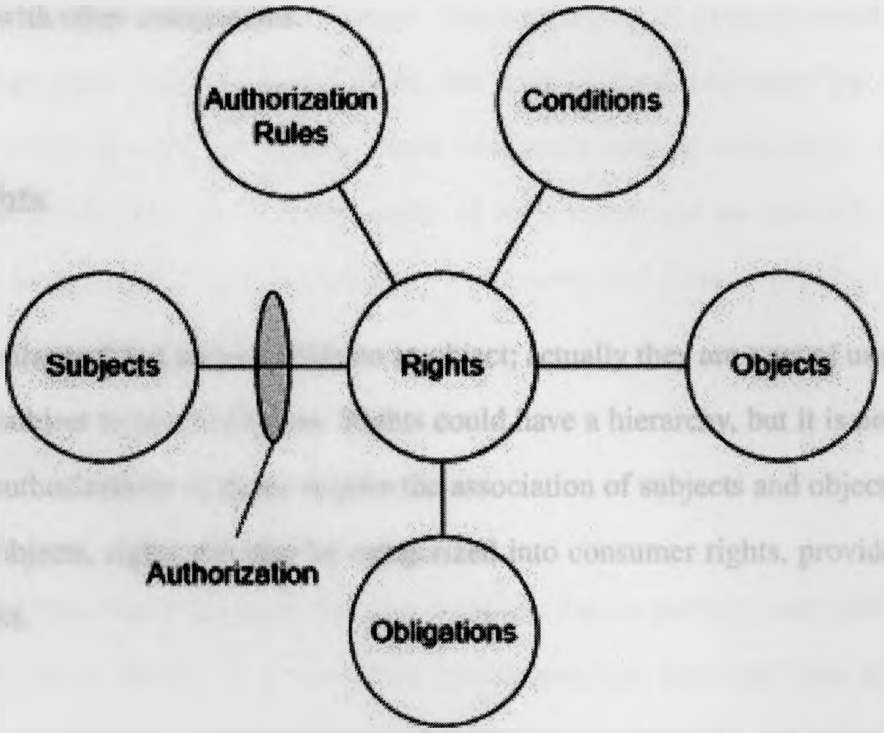

Figure 2.3: UCON model components (From [7])

identifier subjects are entities which are identified in objects that include their privacy-sensitive information. A patient in a health care system is an example of an identifier subject.

2.3.2 Objects

Objects are entities which subjects hold rights on; the subjects can access or use objects. In UCON, objects can be either privacy sensitive or privacy non-sensitive [7]. A privacy-sensitive object includes information which could be used to identify an individual. This kind of object can cause privacy problems if not used properly. A UCON object can also be either original or derivative. The derivative object is an object that is created as a result of obtaining or exercising rights on an original object. For example, opening of a document can create usage log information. This log data file is called a derivative object in UCON. Like the original object, this derivative object is also considered as an object and also holds UCON properties

and relations with other components.

2.3.3 Rights

Rights are privileges that a subject holds on an object; actually they are a set of usage functions that enable a subject to access objects. Rights could have a hierarchy, but it is not required in UCON. The authorizations of rights require the association of subjects and objects. Similar to subjects and objects, rights can also be categorized into consumer rights, provider rights and identifier rights.

2.3.4 Authorizations

Authorizations are a set of requirements that should be satisfied before allowing subjects' access to or use of objects. There are two kinds of authorization rules, the Rights-related Authorization Rules (RAR) and Obligation-related Authorization Rules (OAR). The RAR is used to check if a subject has enough privilege to exercise certain rights on an object. The OAR is used to check whether a subject has agreed on the fulfillment of an obligation that has to be done after obtaining or exercising rights on an object.

2.3.5 Conditions

Conditions are a set of decision factors that the system needs to verify at authorization phase before allowing usage of rights on a digital object. There are two types of conditions: dynamic conditions and static conditions. Dynamic conditions include information that may have to

be checked for updates at each time of usage. Some examples of dynamic conditions are the number of usage times (e.g., can read 5 times, can print 2 times), and usage log (e.g., already read portion cannot be accessed again). Static conditions include information that does not have to be checked for updates. Some examples of static conditions are accessible time period (e.g., business hours), accessible location (e.g., workplace), and allowed printer name.

2.3.6 Obligations

Obligations are mandatory requirements that a subject has to perform after obtaining or exercising rights on an object. In a real world implementation, this may have to be done by agreeing on the fulfillment of obligations before getting the rights and at the time obligation-related authorization rules are checked. For example, a consumer subject may have to accept some payment agreements before obtaining the rights for the usage of certain digital information, or one may need to agree on providing usage log information to a provider subject before reading an e-book or listening to a music file. Traditional access control does not recognize the obligation concept. Recent Digital rights management (DRM) solutions are likely to include obligation functions though many of them implement the obligation functions only partially and implicitly.

2.4 UCON Enforcement Mechanisms

The UCON model and its core components have been introduced above. In this section another issue will be discussed: how the UCON model is mapped into real world computer systems. There are some security mechanisms [8] implemented within a computer system used to achieve the UCON security goals.

2.4.1 Digital Container

A digital container is a cryptographic carrier of digital information that uses encryption, a digital signature or digital certificates to ensure data confidentiality and integrity [9][10]. A UCON system uses this kind of technology as the key element to prevent unauthorized accesses to the protected digital content. It could be implemented in almost any kind of computer system.

2.4.2 Digital Watermarking

A watermark is a small amount of data inserted into a digital object for a variety of purposes. In UCON, it is usually used to enable the tracking of the redistribution of digital objects. Watermarking is sensitive to the type and size of the digital object. Different types of content need different watermarking technologies; also the size of the contents should be large enough to hold the watermark.

2.4.3 Tamper Resistance

A digital object embedded in a digital container is only accessible on the client side using specialized software, e.g. a virtual machine. The virtual machine enforces the usage control policy and is executed in the subject's environment (which is possibly hostile and untrusted). Tamper resistance systems protect the trusted software (e.g., the virtual machine) running on the malicious host. Both software based and hardware based tamper resistant approaches exist in real world applications.

2.5 Implementation

In this section, some examples of real world usage control implementation will be introduced.

2.5.1 Operating System

Nowadays, more and more so-called kernel level attacks happen. So the protection of the kernel integrity becomes one of the most essential security objectives in building a trustworthy OS. Several papers have proposed simple and effective approaches based on the UCON model for the Linux kernel [11][12][13]. In one model called the $UCON_{ki}$, test results show it is capable to detect intrusions and prevent malicious activities by intercepting events in realtime. The model was successfully tested with 18 real-world kernel-level rootkits compromising the OS kernel integrity.

2.5.2 Collaborative Computing System

A collaborative computing system is the most promising area of applicability of the usage control model. Several attempts have been made to apply the UCON model to collaborative computing systems. In [14] the authors describe a formal model for usage control in GRID systems based on a process-based policy language. In their initial work, the framework is very generic although the implementation is for the Globus toolkit [15].

2.5.3 Others

Besides the systems described above, the UCON model can be adapted and implemented in various computer systems and environments, like in service-oriented platforms [16][17], cloud computing platforms and mobile computing environments [18].

2.6 Delegation and Revocation of Obligation

Previous policy could only support the delegation and revocation of authorizations but not obligations. Schaad proposed policies for delegation [19] and revocation [20] for obligations.

The motivations for the delegation of obligation can be summarized as the lack of resources, competence, specialization and organizational policies. When an obligation is delegated between two subjects, the general intent of the delegating subject is to make the receiving subject perform a set of actions. In general, an obligation must be held by a single subject in order to ensure that tasks are only carried out once. This implies that after the delegation took place the delegating subject will no longer hold the obligation object. Similar to the delegation of authority, obligations are usually delegated downwards along a management chain but in certain cases (e.g. illness of an employee) an obligation might be delegated from a subordinate to his superior. Likewise, a horizontal delegation of an obligation can occur.

Revocation of an object is based on its previous delegation, so the following information is needed:

1. The principals involved in previous delegation.

2. The time of the previous delegation.

3. The object and subject of previous delegation.

One thing that needs to be pointed out is that propagation may apply as a principal is able to delegate a delegated obligation. In this case, the principal should only be able to revoke a delegated obligation from the principal he/she delegated it to.

2.7 Negotiation

When making a decision on authorization, traditional authorization systems require some explicit notion of the users accessing the resources provided by the system, like a password or some other digital credential. However as the Internet develops, many resources or services are provided through open systems such as the World Wide Web or peer to peer networks. In these systems, it is not possible that entities will have explicit knowledge of the peers that they are communicating with. So in the context of large scale open systems, authorization decisions are best made based on the attributes of the users in the systems.

Trust negotiation [21] is a technique developed to allow peers to conduct bilateral and iterative exchanges of digital credentials to bootstrap trust relationships in open systems. Figure 2.4 shows an example of trust negotiation.

In [23], the authors proposed Traust, a general purpose authorization service based on trust negotiation. It could provide a uniform interface for clients to get the credentials necessary to access resources provided by systems in a different security domain and acts as a viable migration path for the adoption of trust negotiation research into existing open systems.

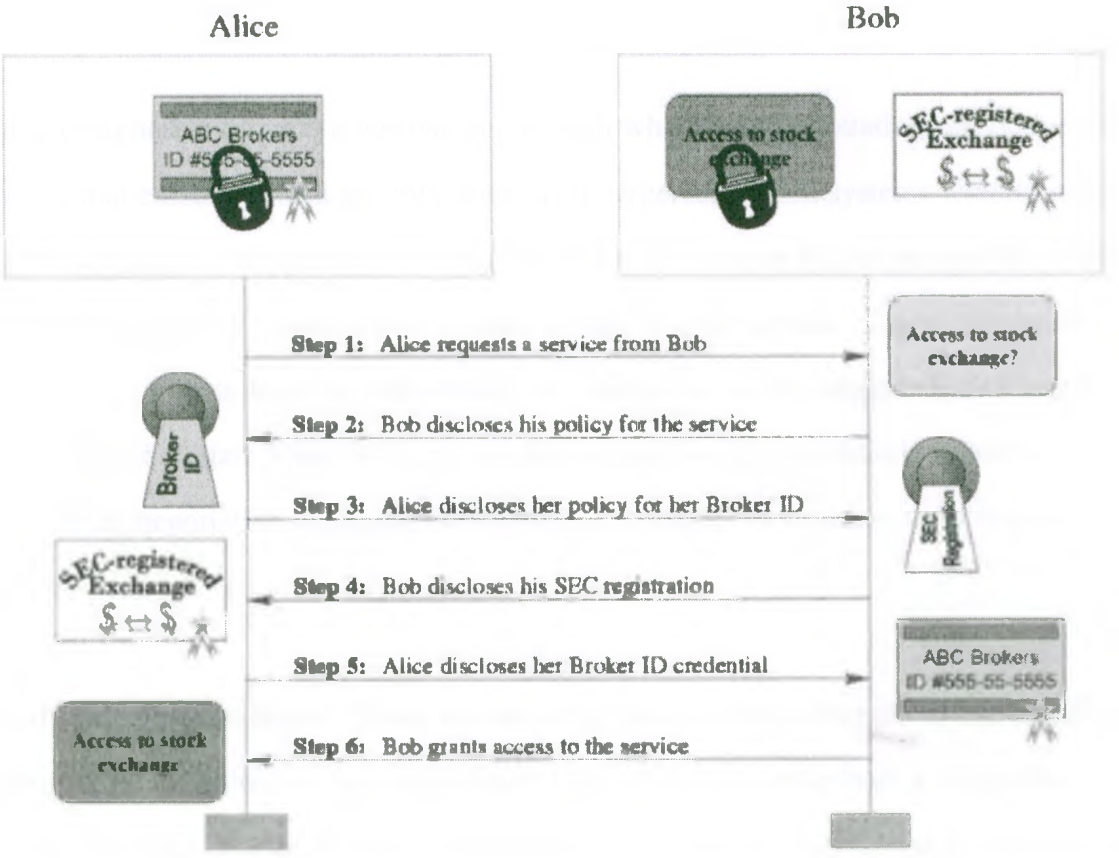

Figure 2.4: Example of trust negotiation (From [22])

The design of Traust embodies five major design goals. These goals help Traust act as a scalable and flexible authorization service for large-scale open systems:

1. Bilateral trust establishment

2. Run time access policy discovery

3. Privacy preservation

4. Support for legacy and trust-aware applications

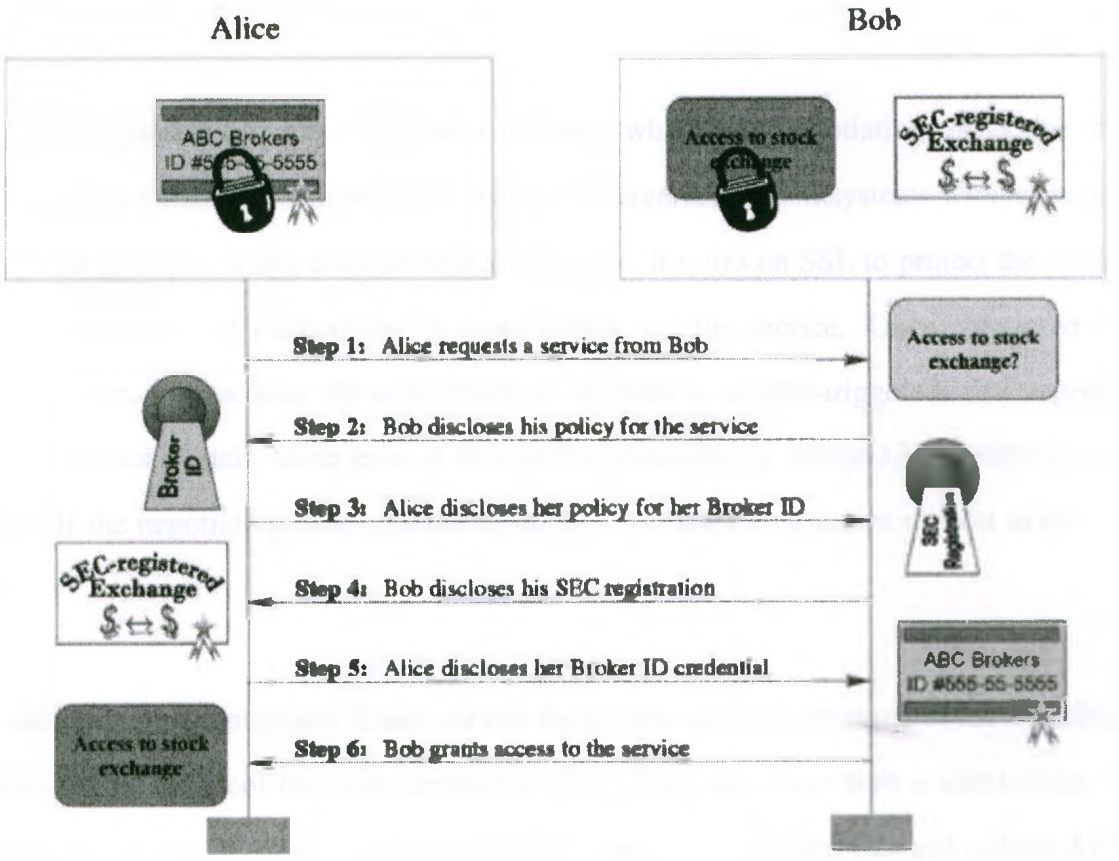

Figure 2.4: Example of trust negotiation (From [22])

The design of Traust embodies five major design goals. These goals help Traust act as a scalable and flexible authorization service for large-scale open systems:

1. Bilateral trust establishment

2. Run time access policy discovery

3. Privacy preservation

4. Support for legacy and trust-aware applications

5. Light-weight, yet robust

Traust is designed to provide a mechanism through which trust negotiation can bridge the security gap that exists between security domains in large-scale open systems without requiring widespread protocol or application software updates. It relies on SSL to protect the confidentiality and integrity of connections between clients and the service. Once connected to the Traust service, clients have the opportunity to conduct a content-triggered trust negotiation with the service to gain some level of trust before disclosing a potentially sensitive resource request. If the negotiation succeeds, the client then discloses its resource request to the Traust server.

The authors implemented one Traust service using Java and the leverages of the TrustBuilder framework and protocol for trust negotiation [24]. They also have both a stand-alone client application that can be used to obtain credentials to access legacy services and a client API that can be incorporated into the design of trust-aware applications.

2.8 Summary

In this chapter, some related work to this thesis has been reviewed. After givings the definition of control domain and reference monitor, the UCON model components were introduced in detail. Then we talked about the UCON enforcement mechanisms and real world implementation of UCON. Finally the delegation and revocation of obligations and negotiation have been discussed.

Chapter 3

Scenarios

In this chapter, two scenarios used for proof of concept of the model are introduced, one for an open system and the other for a closed system. A system is commonly defined as a group of interacting units or elements that have a common purpose. The units or elements of a system can be people, computers and so on. Firstly the definition of these systems is given; following that is a detailed description of the two scenarios used in our Proof-of-Concept of the model.

3.1 Closed System

A closed system is a system which cannot exchange matter with its surroundings. A schematic representation of a closed system is shown in Figure 3.1. A closed system is the one in which the users or devices which can get access to the resources of the system are pre-defined. So in a closed system, the service or resource provider has a list of users who may access the service or resource. Any access attempt from a user not in the list will be denied immediately.

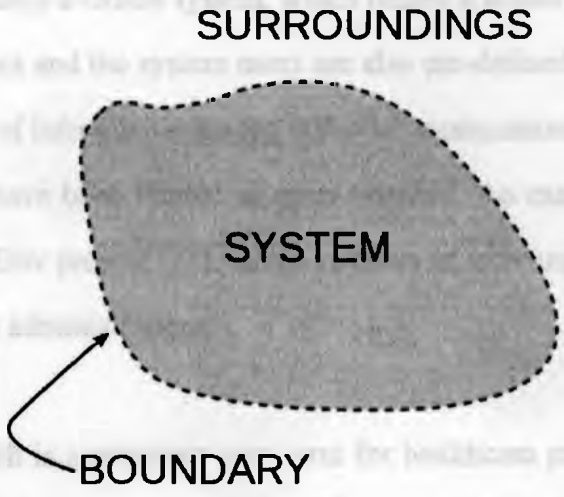

Figure 3.1: A schematic representation of a closed system and its boundary (From [25])

This is the kind of system which has been widely used and studied before. Traditionally, most computer systems are closed ones. Several examples of closed systems are:

Computer System: In a traditional computing environment, if a user wants to access the computing resource, he/she has to have a valid account or username on the system. The system maintains a list of users and corresponding privileges the user has on the system. Before accessing resources, a user must provide his/her account information to the system for authorization. After the user successfully logs into the system, he/she may send access requests to the server side, then the system will check to see if the user has the privilege to access the particular resource or service. Taking a Unix/Linux system as an example, all the usernames are stored in file */bin/passwd*, and by using a DAC model, each file in the system has a property indicating the rights each user has on the file.

eGovernment Workflow System: Workflow automates the management and coordination of organizational or business processes. Electronic government (eGovernment) is the civil and political conduct of government, including services provision, using information and communication technologies [26]. Because of the privacy requirement of this kind

of system, it is usually a closed system, which means it doesn't need to interact with the outside environment and the system users are also pre-defined. Recently there has been a need for sharing of information among different organizations, so in some work, eGovernment systems have been treated as open systems. An example of an eGovernment system is the R4eGov project [27], which consists of inter-organizational collaboration between European administrations.

eHealth System: eHealth is a relatively new term for healthcare practice supported by electronic processes and communication [28]. The main idea of eHealth is to use the power of information technology to improve patient care. It may include a range of systems or services like electronic health records, telemedicine, consumer health informatics, health knowledge management, *etc.*. Here I mostly focus on the electronic health records part. Only a small range of certain people (like doctors, nurses, pharmacists, *etc.*) have access to patients' health records and most of the time this access right is not unlimited. Thus this is also a closed system.

Closed system is one kind of typical system. When testing our model, it has to be covered. In the proof of concept later, we will use an eHealth system scenario to demonstrate the model. A detailed description of the eHealth scenario will be given in a later section.

3.2 Open System

Besides closed systems, there is another kind of mainstream system, the open system. Unlike closed systems, open systems refer to systems that interact with other systems or the outside environment. The schematic representations of open systems are shown in Figure 3.2 and 3.3. In our work, open system mainly refers to a system in which the users or devices that can

access the system are not pre-defined.

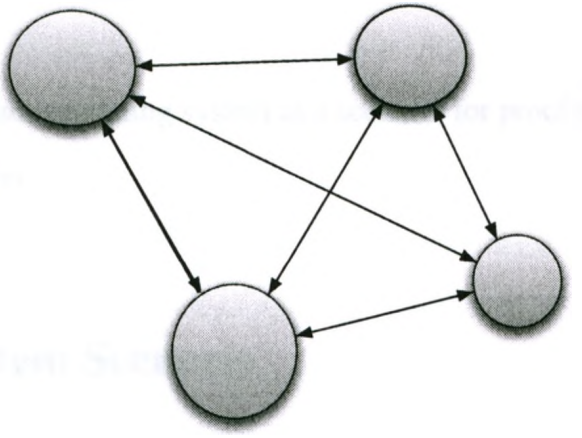

Figure 3.2: A schematic representation of a open system (Interacting with other systems)

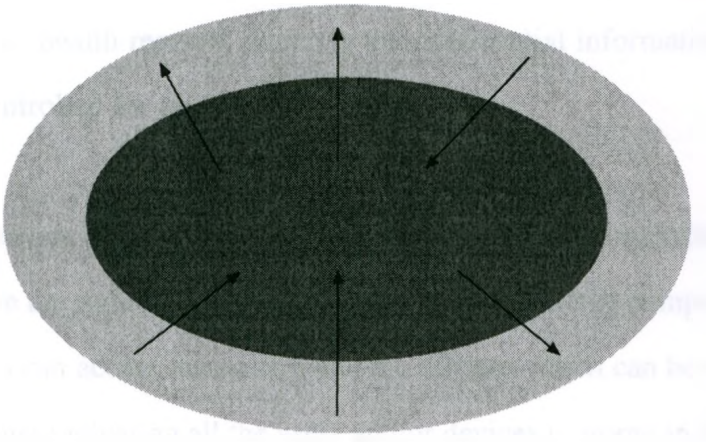

Figure 3.3: A schematic representation of a open system (Interacting with outside environment)

Open systems is a relatively new idea. Due to the rapid increase of applications, it has drawn a lot of research attention. Some examples of an open system might be a cloud computing system (like Amazon EC2 [29]), collaborative computing systems, *etc.*. In these systems, there are no pre-defined, fixed number of users. Anyone can get a valid account for the system simply by registering or paying for the service. The server of the system doesn't have a list of

valid users; in fact, the list of users doesn't exist at all. Due to this, the control of the system is more difficult than in a closed system.

We will use a typical cloud computing system as a scenario for proof-of-concept purposes; the scenario will be given later.

3.3 Closed System Scenario

3.3.1 Scenario

Here we consider an eHealth system as the scenario for a closed system. More specifically, we will focus on electronic health records, which is the confidential information for patients and needs to be strictly controlled for access.

The patients' health records are stored in the server in a hospital. A person (like doctor, pharmacist or nurse) may have the right to access the records using a desktop computer or a hand-held device. The users who can access the system and the devices which can be used for accessing are pre-defined and a list indicating all the users and/or devices is stored in the server.

Only authorized actors with recognized certificates are allowed to access patient records. Access rights are based on the roles of the actors. Even for a valid user, he/she cannot access all the records without limitation. For example, pharmacists are allowed only to access the part of the healthcare record containing prescriptions. Doctors can only access the records of their own patients.

In case of emergency, the Break-Glass Policy [30] has to be considered. Access control models

are usually static, for example permissions are granted based on a policy that seldom changes. Especially for scenarios in health care and disaster management, a more flexible support of access control is needed. Break-glass is one approach for such a flexible support of policies which helps to prevent system stagnation that could harm lives or otherwise result in losses. Here in my scenario, a Break-Glass Policy is defined as an authorization scheme to allow access to patients' medical records in case of emergency. An attending physician may need to bypass routine access control restrictions to guarantee timely treatment without any delay due to administrative or technical complexities. However, this policy must be controlled correctly to prevent someone from misusing it to get patients' private information.

A user can delegate his/her rights to other user(s) in specified situations with defined limitations. For example, in case of a patient referral, the primary physician can delegate his rights to the specialist or the patient can grant access to the specialist using the delegated rights of the primary physician. Delegation of rights is also needed when the physician is not available and delegates his rights (for some duration, or for some specific purpose) to another physician/nurse to continue the the patient's treatment.

The 4-Eyes-Principle [31] has to be followed. The 4-Eyes-Principle is a form of Multiple Authorization, which requires two users with a common interest to enter the system simultaneously. One of the users accesses the data while the other user monitors the access in order to ensure data confidentiality and integrity during access. In healthcare scenarios, the 4-Eyes-Principle requires the patient to be present, when a physician accesses the patient's medical record.

A patient record should be saved in the doctor's machine for a maximum period of time. After this specified time, the record should be deleted from the doctor's machine. This is called the retention time.

The doctor should only be able to use the patient's record within the hospital, for example, if the record is in the doctor's desktop computer, he must not be able to copy it out with a flash drive, or the copy cannot be viewed with another machine. If the record is in a hand-held device like PDA or iPhone, the access must be restricted when this device is not in the range of the hospital.

After the end of a treating session, the retrieved document could be stored in the local machine of the doctor in case the patient approves it, otherwise it should be deleted from the doctor's system.

In case the patient is not present before the normal termination of a treatment session, the document must be deleted and an abnormal session notification should be reported to the service provider.

3.3.2 Obligations

Based on the scenario discussed above, all obligations appearing in the scenario are summarized below.

Pre-obligations

In order to get access to the patients' records, the actor must provide enough information to the system to get authorized. This obligation is somehow overlapped with the authorization process.

Ongoing-obligations

One person can only access the information he/she is allowed. In case of emergency, when the break-glass policy is applied, the actor still needs to provide information showing the situation.

When delegating rights to others, it is the actor's obligation to check the person's privileges.

The 4-eyes-principle has to be met when accessing sensitive information.

The actor must not be able to take the records out of the control of the system.

Post-obligations

After the treatment session, the patient's record can be saved in the doctor's machine for a period of time; after this time, the record must be deleted.

In case the patient is not present before the normal termination of a treatment session, the document must be deleted and an abnormal session notification should be reported to the service provider.

3.4 Open System Scenario

3.4.1 Scenario

For an open system scenario, we consider a cloud computing service.

A cloud computing system is an Internet-based computing system, where shared resources,

software, and information are provided to computers and other devices on demand [32]. The cloud is the network which is constructed through the cloud computing model, and a cloud service is the service provided in the cloud. Now, Cloud Computing has become the hottest technology in IT, and is also a research focus in academia. Figure 3.4 shows a conceptual diagram of a cloud computing environment.

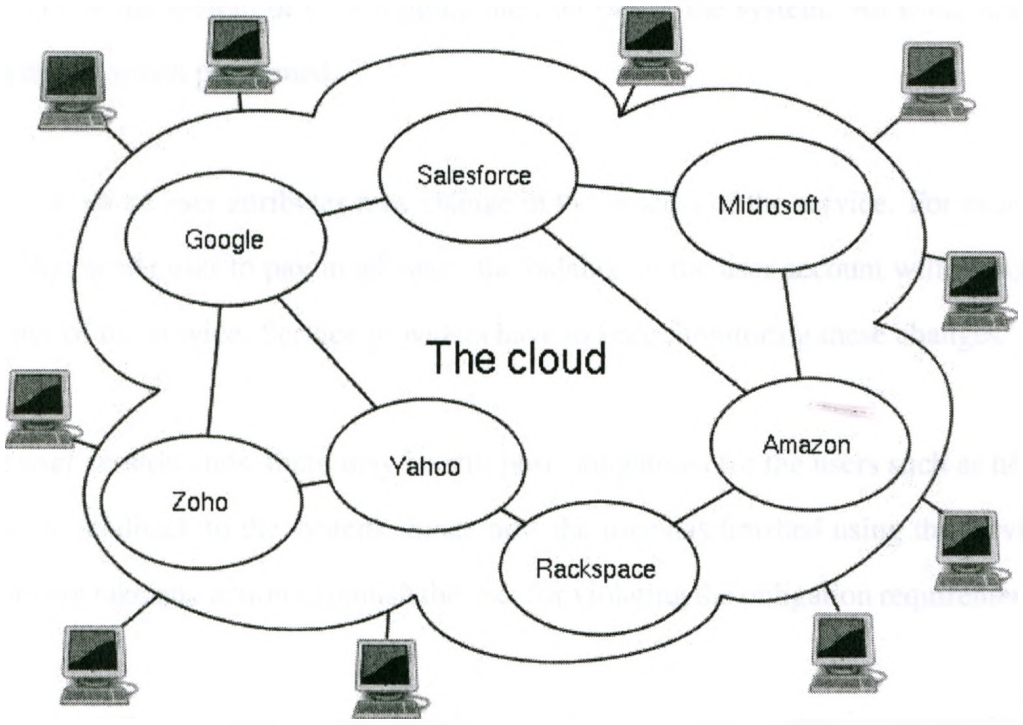

Figure 3.4: Cloud computing conceptual diagram (From [33])

A cloud service is based on web services, and web services are based on the Internet. The Internet has many security flaws because of its openness. Therefore, cloud services will face a wide range of security issues. Cloud services have several features like large amounts of resources, highly dynamic and flexible construction, lots of dynamic users and so on. Because the users of the system are not pre-defined, it is an open system.

The cloud service system provides services to end users. When a user needs service from the system, first he/she needs to get authorization. Authorization is based on user attributes. Besides that, a user may have to agree to follow some obligations in order to get authorized to

the system.

After getting authorization, when a user is using the services provided by the system, the system monitors the user's behaviors to see if the obligation is met. Once the obligation is not met, the system could take action right away. Possible actions include restricting the access to some of the resources in the system or even logging the user out of the system. An event log may be another kind of action performed.

What's more, some user attributes may change in the process of the service. For example, for a service that needs user to pay in advance, the balance of the user account will change along with the use of the service. Service providers have to keep monitoring these changes.

After the user session ends, there may be still post-obligations for the users such as he/she has to give some feedback to the system. Since now the user has finished using the services, the system can not take any action to punish the user for violating the obligation requirement.

Because a user may still need the service later on, when a post-obligation is not met, the system may still have to perform some actions to log this event in local system for later evaluation purposes.

Sometimes the user maybe not so familiar with the requirements of the system. He/she may not provide enough information for the service provider to grant the user proper rights. The attributes are insufficient or the condition parameters are inconsistent. When this happens, it is the service provider's decision what to do with the request: either deny the request immediately or go to some kind of negotiation process.

3.4.2 Obligations

Based on the scenario discussed above, all obligations appearing in the scenario are summarized below.

Pre-Obligations

In order to get access to the service from the system, the user needs to get authorization first. For authorization purposes, user needs to provide his/her attributes required.

User needs to accept some obligations or terms of services before being authorized.

Ongoing-Obligations

When using the service of the system, the system monitors the user's behaviors to see if the obligations are met. Besides that, the system will also monitor the user for abnormal access requests .

For some user attributes which will change during the access session, the system tracks the changes and the users may need to adjust their accessing behaviors according to these changes.

Post-Obligations

Users may need to give feedback to the system.

If the service is not pre-paid, after the usage session ends the users need to pay for the service.

3.5 Summary

In this chapter, the details about the scenarios which will be needed for the proof of concept of our model were explained. Those two kinds of system scenarios, namely the closed system and open system, together could represent almost all the systems deployed in real world applications. In a later chapter the model proposed will be applied to these two scenarios for demonstration purposes.

Chapter 4

Obligation Model

In this chapter, a new obligation model for usage control will be proposed. First the existing obligation model in UCON is reviewed. After that the new obligation model will be explained, which is an improvement of the original one.

4.1 Original Obligation Model

4.1.1 Obligation Model in UCON$_{ABC}$ Model

Sandhu and Park [34] proposed the initial work on usage control, the UCON$_{ABC}$ model. They gave the definition of obligation as the requirement that a subject must perform some action before or during access. Based on the definition, the obligation model has two parts, namely the Pre-obligations Model (UCON$_{preB}$) and Ongoing-obligations Model (UCON$_{onB}$).

Pre-obligations Model

Pre-obligations stand for the obligations that have to be fulfilled before access permission is granted. Due to this, the pre-obligation sometimes overlaps with authorization. Some examples of pre-obligation include requiring a user to provide a valid E-Mail address before using the service or asking the user to choose to accept a term of service in order to get registered in the system.

Ongoing-obligation Model

Ongoing-obligations are the obligations to be fulfilled while rights are exercised. There are basically two kinds of ongoing obligations, namely the ones to be fulfilled periodically and the ones to be fulfilled continuously. One example of periodically-fulfilled ongoing-obligation is asking the user to update the file he/she is current accessing at least every 10 minutes. An example of a continuously-fulfilled ongoing-obligation is requiring the user to keep an internet connection alive while accessing a document.

UCON state transition

In [35] Zhang *et al.* gave a formal model and specification for UCON_{ABC}. Figure 4.1 shows the state transition diagram of the system. In a single usage process, six states have been defined, namely *initial*, *requesting*, *accessing*, *denied*, *revoked* and *end*. *Initial* is the state in which the access request has not been generated. *Requesting* is the state when the user is waiting for the system's decision for his/her access request. *Accessing* means the request has been approved and the user is using the service/document. *Denied* indicates the state in which the access

request is not approved by the system. The *end* state is the one in which the user ends the access normally and *revoked* indicates the end of access by the system.

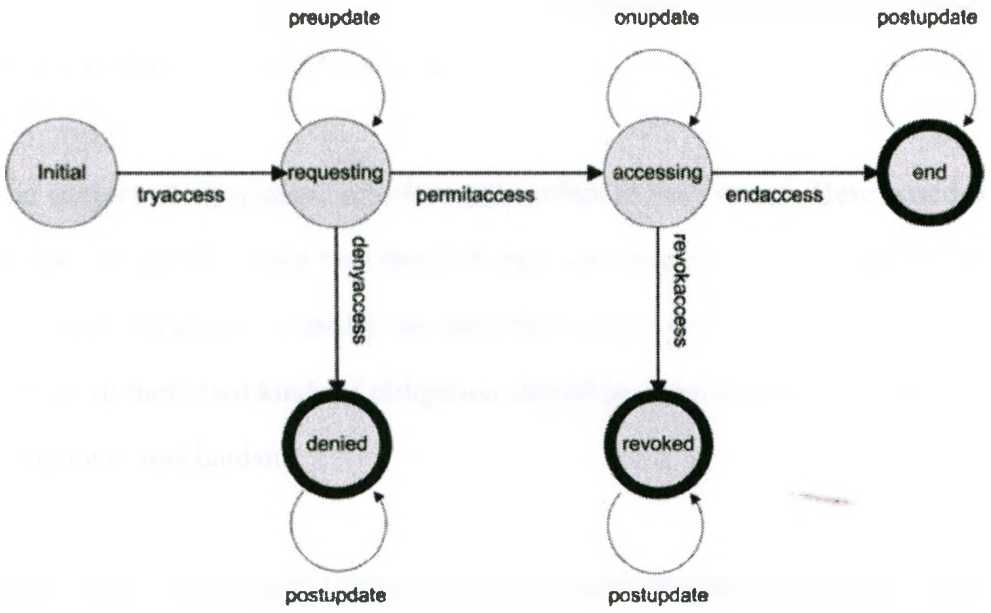

Figure 4.1: Original UCON state transition diagram (From [36])

4.1.2 Enhanced Obligation Model

The main drawback of the original obligation model in the last section is the inability to handle obligations after the usage session ends. So later in [36] the authors tackled this problem by introducing post-obligations to the UCON model. The purpose of a post-obligation is twofold: 1) It can be used to execute obligation actions that are related to the current usage despite the fact that it has no affect on the decision making of the current usage. 2) It can affect future usage sessions. Along with the model is the proposition of a new state transition scheme.

Obligation Model

An obligation can be expressed as a tuple, OBL = (OBS,OBO,OBA,WHEN,DURATION). The meaning of each element is described as follows:

OBS is the subject of obligation, which usually refers to the system. Here based on the subject, the obligation could be classified into system-performed obligation and subject-performed obligation. Usually we have full control over the system but not the other subjects, so these two kinds of obligation should be treated separately when comes to the enforcement mechanism.

OBO is the object of obligation. Objects can be treated as the ones the system controls and the ones out of the control of the system, namely controllable objects and non-controllable objects.

OBA is the action which is going to be performed. For example to read a file, to use a service or to delete a document from the system.

WHEN refers the type of obligation (*pre|on|post*). It is very important to separate the obligations after the usage session ends (post-obligation) from the ones before the usage session ends. This is because after the usage session ends, whether the user fulfilled the obligation has no effect on the usage session any more while it does during or before the usage session.

DURATION is the period or time point to check the fulfillment of the obligation. That means the obligation should be fulfilled within a period of time.

State Transition

The original UCON state transition diagram does not show the actions triggering the ongoing decision check during the usage session. The accessing state and the state in which ongoing decisions are checked are merged into one state, the accessing state. So in the enhanced obligation model the authors gave an extended UCON state transition scheme, shown in Figure 4.2.

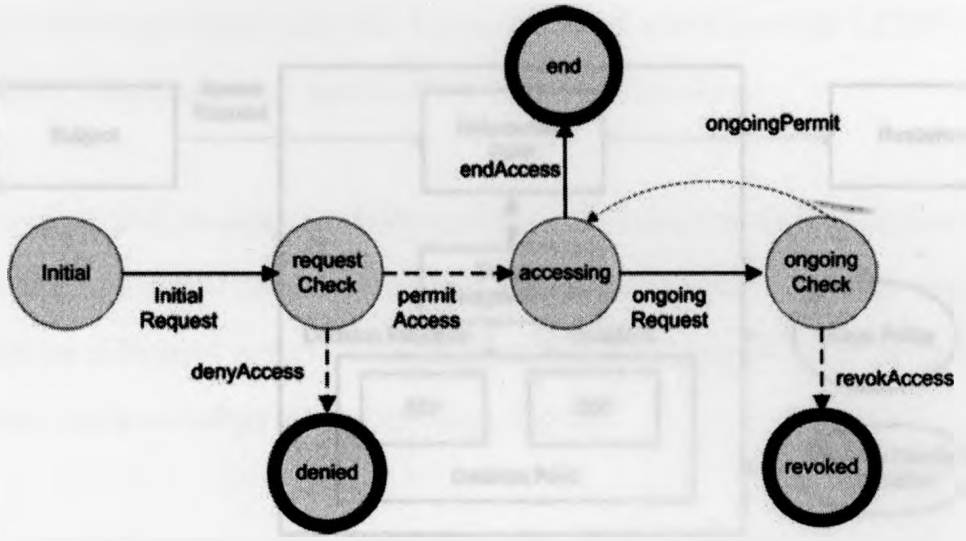

Figure 4.2: Expanded UCON state transition diagram (From [36])

Compared with the original state transition scheme, the expanded one omits the update actions (*preupdate*, *onupdate* and *postupdate*) because the updates are considered as system obligations. Also a new state, the *ongoingCheck* state and two new transitions, *ongoingRequest* and *ongoingPermit* are added in this expanded scheme. In this new scheme, any changes of attributes will trigger the *ongoingRequest* transition and the system will move to the *ongoingCheck* state. Any updates occurring in the *ongoingCheck* state will have no direct effect on the current evaluation but will be checked in *accessing* or *revoked* states later.

Enforcement Model

The enforcement model of UCON has three core components: *enforcement point* (EP), *decision point* (DP) and *session management point* (SMP), as shown in Figure 4.3. When the system receives an access request, EP forwards it to SMP. Then the SMP sends pre-decision request to DP. Based on the usage policy and other needed information, DP returns the final decision back to SMP. Ultimately EP gives the decision to the end user.

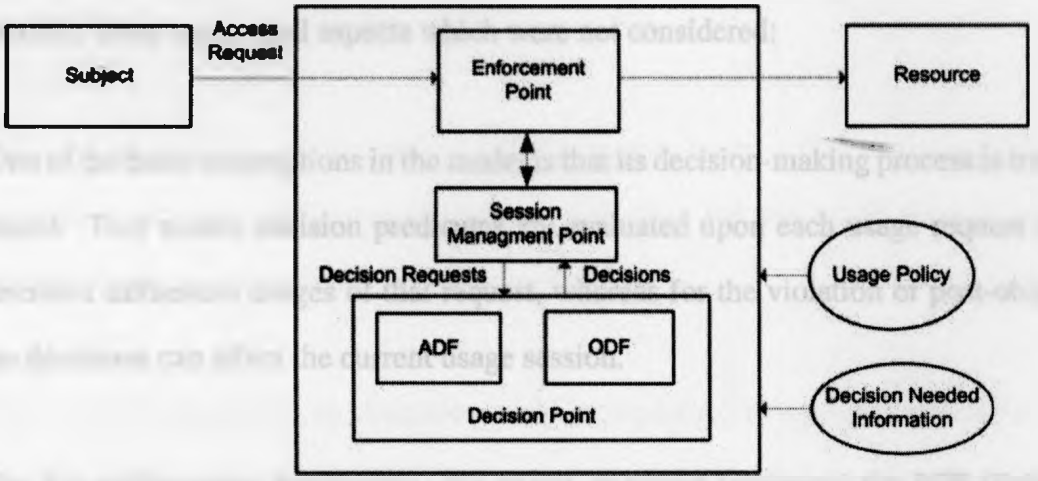

Figure 4.3: Usage Control Enforcement Model (From [36])

DP has two sub-models as follows:

ADF The Attribute Decision Function is responsible for all the attribute-based access decision during an access request. Attributes can be on subjects, objects or can be environmental.

ODF The Obligation Decision Function handles decisions related to obligations. It checks whether an obligation has been fulfilled or not.

4.2 New UCON Model Focusing on Obligation

4.2.1 Motivation

In this section, a new model based on the ones described above will be proposed. Although in [36] the authors introduced the idea of post-obligation which made the UCON model much more flexible, there are several aspects which were not considered:

1. One of the basic assumptions in the model is that its decision-making process is transaction-based. That means decision predicates are evaluated upon each usage request and the decision influences usages of that request, whereas for the violation of post-obligation, no decisions can affect the current usage session.

2. For the enforcement mechanism, the model proposed integrated the PEP (Policy Enforcement Point) into the application. That approach is not flexible. In order to use the object, the user must use the specified application. On the other hand, this could also bring security issues, since the PEP is in the application, so if an alternative application could be provided by a third party, this new application may choose simply to ignore the obligation requirements.

3. The model proposed is only suitable in a closed system. Nowadays more and more open systems are widely used. Unlike closed systems, in an open system the users who require resources are not pre-defined. In order to cover this new application scenario, modification has to be made to the obligation model.

4.2.2 Obligation Model

The meaning of having obligation in UCON is to check whether a certain activity has been fulfilled or not. The pre-obligation and ongoing-obligation have been discussed before. In general the purpose of having a post-obligation is twofold:

1. The post-obligation can be used to execute an obligation action relating to a current usage session although it can no longer affect the decision making of the usage session. However sometimes the service provider needs to know whether these obligation activities are fulfilled.

2. With the introduction of mechanisms like the history-based policy engine which will be described later, the fulfillment of the post-obligation for one usage session can affect decision making processes in future usage sessions.

Similar to the previous model, an obligation could be expressed as a tuple shown before which included five elements, namely the obligation subject, obligation object, obligation action, the type of obligation (pre-obligation, ongoing-obligation or post-obligation) and the duration of the fulfillment of the obligation action. This tuple is the basic unit when defining an obligation.

4.2.3 State Transition

The state transition scheme in the enhanced UCON model cannot deal with post-obligation very well. To be more specific, in that scheme the *end* and *revoked* states are final ones, that means when a usage session gets to any of the two states it simply ends. But in a real scenario,

there are still post-obligations taking effect at that time. To enhance the ability of expressing post-obligations, We propose a new state transition scheme here by adding a new state, the *Exit*, and a *Post check* action.

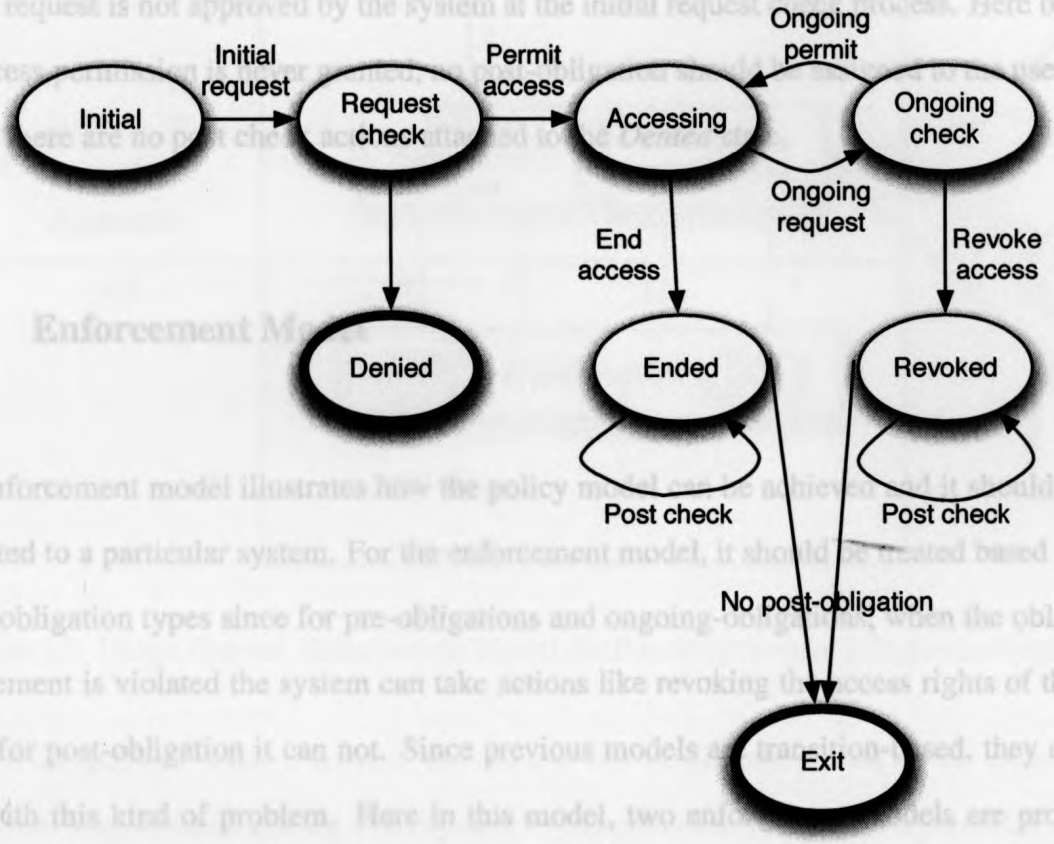

Figure 4.4: Usage Control State Transition Scheme

Figure 4.4 shows our enhanced usage control state transition scheme model. Compared to the state transition shown in Figure 4.2, We add a new state *Exit* as the final state, also two *Post check* trigger actions are added to states *Ended* and *Revoked* respectively which made the two states non-final.

In this new state transition scheme, when access is ended by the user or revoked by the system, the whole access session is not terminated. To ensure the post-obligation will be fulfilled there will still be post checks taking place in these two states until all post-obligations are fulfilled.

When post-obligations are fulfilled and there are no more of them, the system goes to the *Exit* state, which is the final state entered after the *Ended* and *Revoked* states. Coming to this means the whole current access session is ended. *Denied* is another final state which indicates the access request is not approved by the system at the initial request check process. Here because the access permission is never granted, no post-obligation should be assigned to the user. That is why there are no post check actions attached to the *Denied* state.

4.2.4 Enforcement Model

The enforcement model illustrates how the policy model can be achieved and it should not be restricted to a particular system. For the enforcement model, it should be treated based on different obligation types since for pre-obligations and ongoing-obligations, when the obligation requirement is violated the system can take actions like revoking the access rights of the user while for post-obligation it can not. Since previous models are transition-based, they can not deal with this kind of problem. Here in this model, two enforcement models are proposed, one to deal with pre-obligations and ongoing-obligations and one aiming at the post-obligation problems.

Pre-obligation and Ongoing-obligation Enforcement Model

Figure 4.5 shows the enforcement model for pre-obligation and ongoing-obligation. The enforcement model consists of four main components: *Enforcement point*, *Session management point*, *Decision point* and *Attribute inquiry point*.

The enforcement point is the element which directly interacts with the subject and resource.

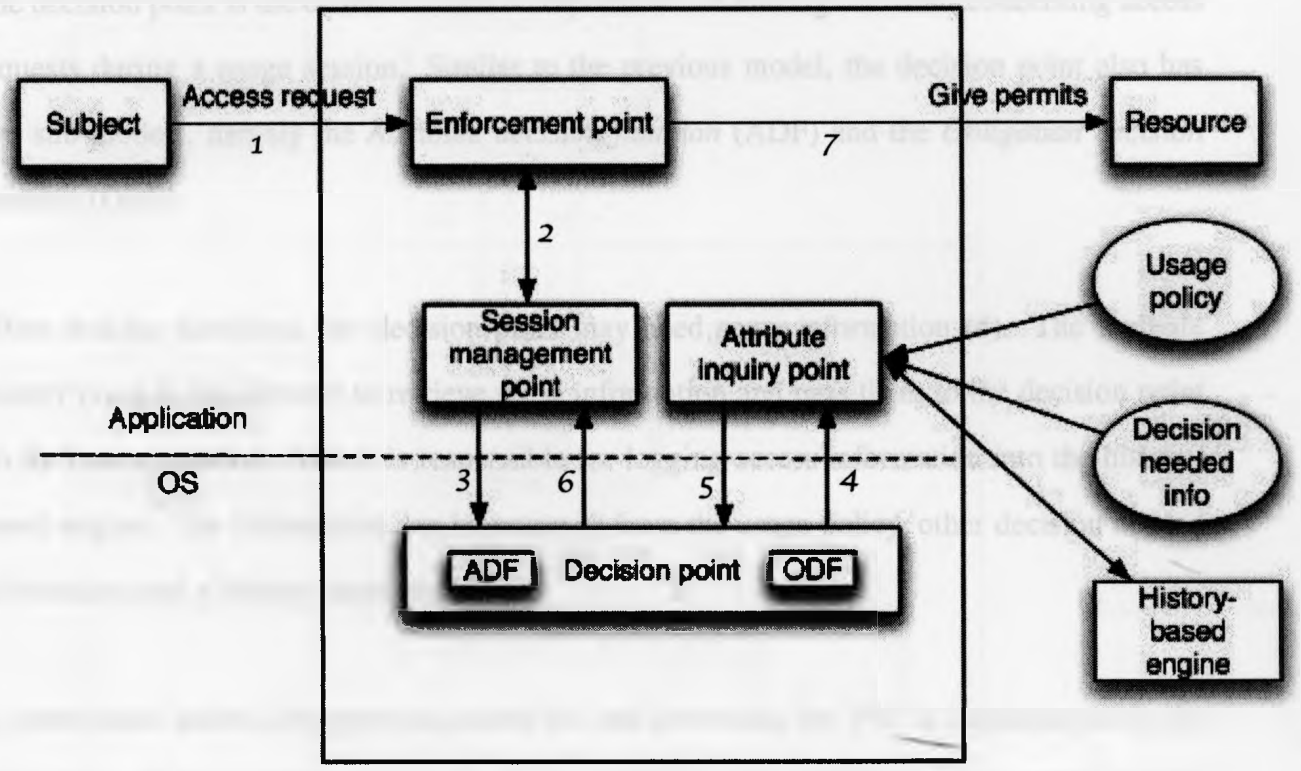

Figure 4.5: Usage Control Enforcement Model for Pre-obligation and Ongoing-obligation

When a subject/user sends a request, the request will be directly passed to the enforcement point shown as the arrow marked 1 in Figure 4.5. The enforcement point then passes the request to other parts of the enforcement model for decision making (Indicated as 2 in Figure 4.5). After the decision has been made, the enforcement point either grants the access permission of the resource (shown in 7) to the subject or informs the subject that its access request is denied.

The session management point acts as an interpreter here that translates the access request received from the enforcement point to a *pre-decision* which can be further processed by the decision point. After passing the *pre-decision* to the decision point (3), if a denied decision is made by the decision point, the session management point sends the decision to the enforcement point (in 6) and also requests the decision point to go to the process of logging the event to the history-based engine; otherwise it sends a permit response to the enforcement point.

The decision point is the element which is responsible for making decisions concerning access requests during a usage session. Similar to the previous model, the decision point also has two sub-models, namely the *Attribute decision function* (ADF) and the *Obligation decision function* (ODF).

When making decisions, the decision point may need some information (4). The attribute inquiry point is the element to retrieve those information and pass them to the decision point (as in 5) as requested. Also it is responsible for logging access information into the history-based engine. The information can be retrieved from the usage policy, other decision needed information and a history-based engine.

As mentioned before, the previous model has one drawback, the PEP is implemented in the application. In this implementation the whole system is vulnerable because the control system is easily bypassed by a self-made application. In this improved model proposed, to tackle this problem, part of the PEP is in a rather low level of the system. To be more specific, the decision point is implemented in the operating system level so that it can not be accessed by normal application and can not be bypassed either. In order to interact with the subject and resources, other elements are at the application level.

Post-obligation Enforcement Model

For post-obligations, unlike pre-obligations and ongoing-obligations, the current usage session is ended when checking the fulfillment of it so no matter what decision is made based on the fulfillment check of the obligation, it can not affect the usage of the resource for the subject. One possible action is to log the violation in the system, so that the next time the subject sends another request, the system will check the history in the decision making process.

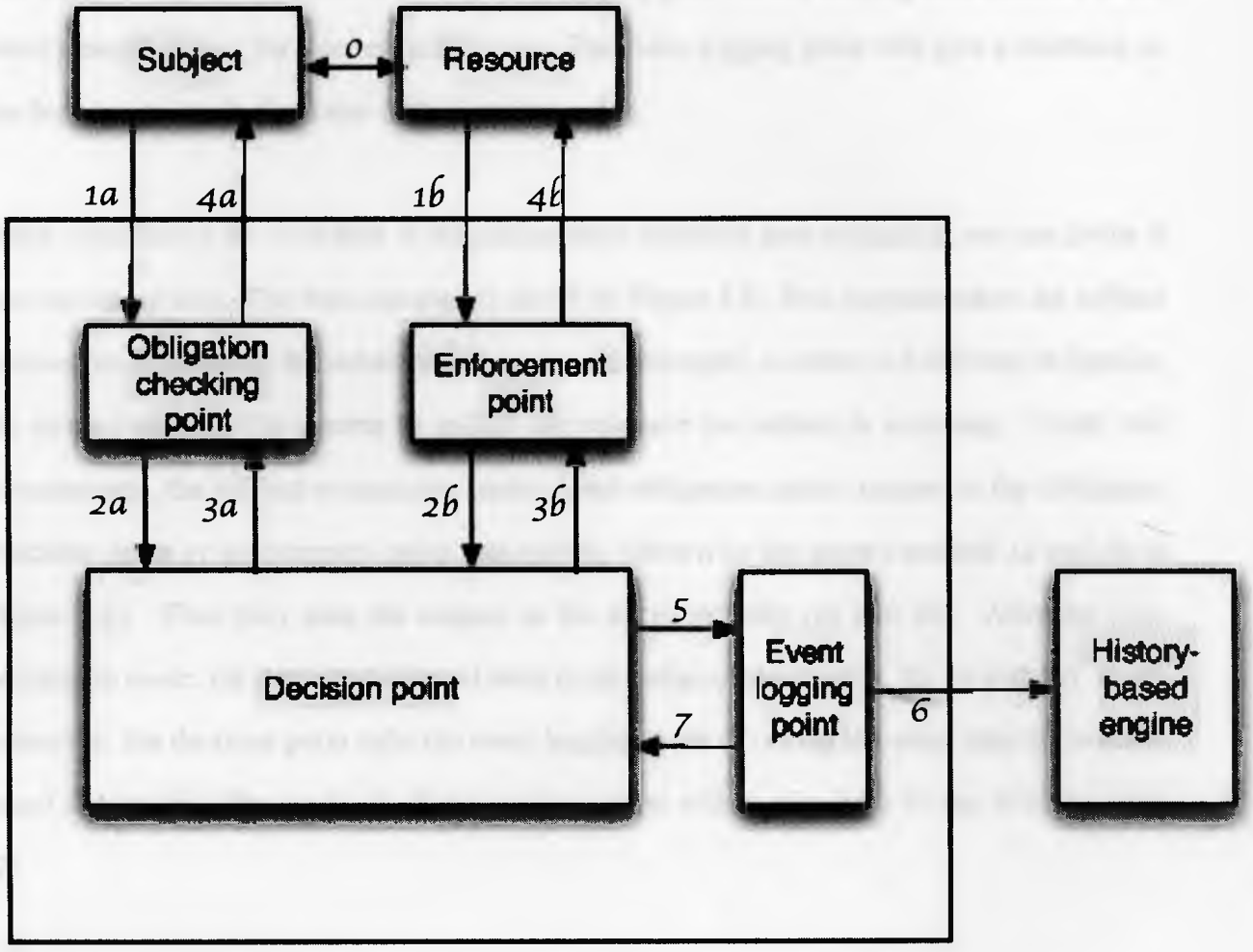

Figure 4.6: Usage Control Enforcement Model for Post-obligation (1)

The post-obligation enforcement model is shown in Figure 4.6. The model consists of four parts, namely the *Obligation checking point*, the *Enforcement point*, the *Decision point* and the *Event logging point*.

The obligation checking point and enforcement point interact with the subject and resource respectively to get their information about the fulfillment of post-obligations and pass the information to the decision point. After the decision is made by the decision point, it passes the decision to the obligation checking point and enforcement point, which further notify the subject and resource about the decision though it can not affect the usage session any more.

The final decision is also passed to the event logging point which will log this decision to the history-based engine for later evaluation use. The event logging point will give a feedback to the decision point on the status of the logging action.

When considering the workflow of the enforcement model of post-obligation, we can divide it into two categories. The first category is shown in Figure 4.6. This happens when the subject or resource is updating the enforcement model. For example, in order to fulfill one obligation, the subject requests the system to update one resource the subject is accessing. Under this circumstance, the subject or resource sends a post-obligation update request to the obligation checking point or enforcement point respectively (shown as the arrows marked 1a and 1b in Figure 4.6). Then they pass the request to the decision point (2a and 2b). After the final decision is made, the decision is passed back to the subject/resource (3a, 3b, 4a and 4b). In the meantime, the decision point calls the event logging point (5) to log the event into the history-based engine (6). The feedback of the logging action will be give back to the decision point (7).

The other case takes place when the decision point starts a query to the subject or resource to check its obligation fulfillment status as shown in Figure 4.7. For example the system checks whether a particular file has been deleted or not after a specific period of time. In this case the first step is the decision point sends a query to the subject or resource through the obligation checking point or enforcement point respectively (shown as the arrows marked 1a, 2a or 1b, 2b *in Figure 4.7). The response is sent to the enforcement point (3b) or obligation checking point* (3a) and further passes back to the decision point (4a or 4b). Based on the feedback received, the decision point calls the event logging point (5) to log the event into the history-based engine (6). After the logging action a feedback from the event logging point is sent to the decision point (7).

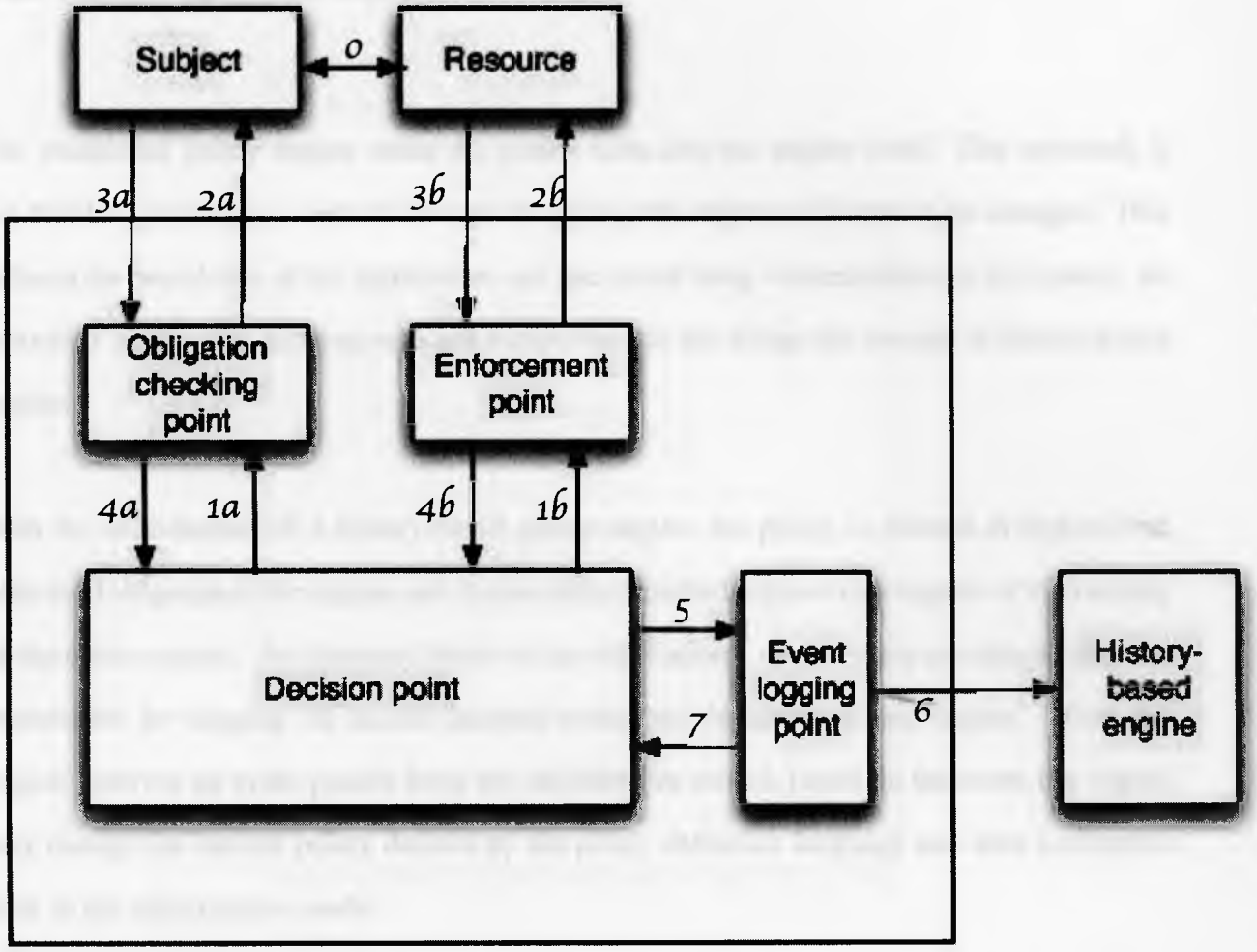

Figure 4.7: Usage Control Enforcement Model for Post-obligation (2)

For post-obligations, the main action when a violation occurs is to log the event in the history-based policy engine and this action is put into effect by the system rather than the subject. Because the system has full control over the action there is no need to put any component of the enforcement mechanism into a lower level for security reasons.

4.2.5 History-based Engine

The traditional policy engine codes the policy rules into the engine itself. This approach is not flexible; if there is a need to change the policy, the engine will need to be changed. This reduces the portability of the application and also could bring vulnerabilities to the system. So in today's application this engine is not acceptable and this brings the concept of history-based engine.

With the introduction of a history-based policy engine, the policy is defined in higher-level definition language in the engine and can be easily modified without interruption of the running of the entire system. As discussed above in the enforcement model, there is a unique element responsible for logging the access decision event into the history-based engine. When the engine receives an event passed from the enforcement model, based on the event the engine may change the current policy defined by the policy definition language and send a response back to the enforcement model.

Figure 4.8 shows the system components of the history-based engine, which mainly consists of three parts, the *Event Bridge*, *Update Point* and *Event Database*.

The *Event Bridge* is the component mainly responsible for directly communicating with the requests sent to the engine. Its job includes accepting the query or updating request from the enforcement point (shown as the arrow marked 1 in Figure 4.8) and passing along the request (2). After getting the response from other component (5), it returns the result to the enforcement point.

The *Update Point* is used as an interpreter. It translates the original request sent from the *Event Bridge* to a command of higher-level definition language and sends it to the *Event Database*

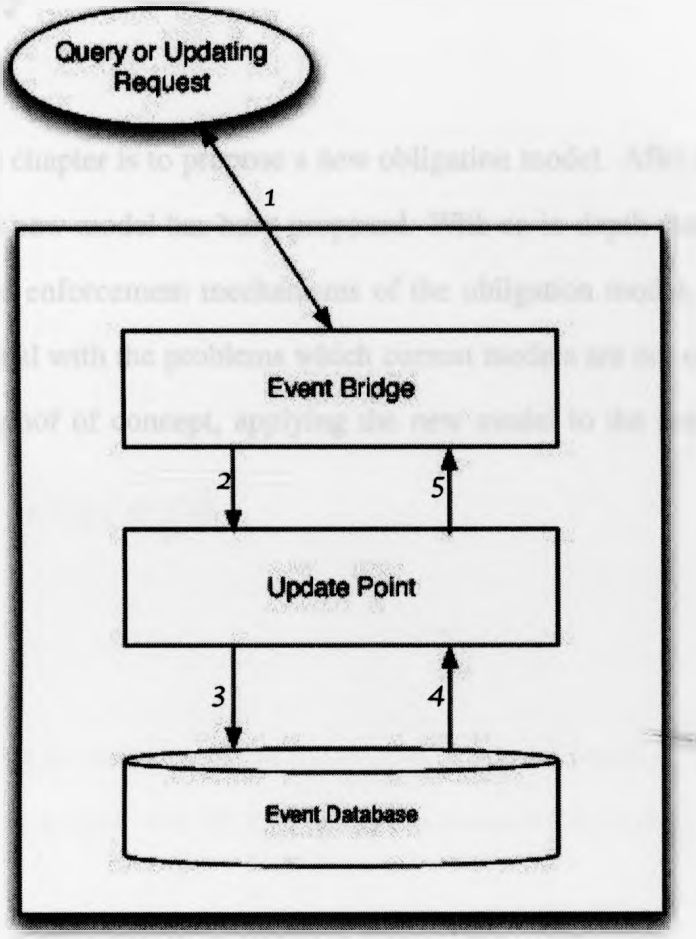

Figure 4.8: History-based Engine

(3). Also after getting a response from the *Event Database* (4), the *Update Point* translates the original response to a format the *Event Bridge* could understand and passes it along.

The *Event Database* is the component which actually stores the policies. Those policies stored here are defined based on the past behaviors of user, or the history.

4.3 Summary

The main idea of this chapter is to propose a new obligation model. After a brief introduction of current models the new model has been proposed. With an in-depth discussion on the state transition scheme and enforcement mechanisms of the obligation model, it can be seen that this new model can deal with the problems which current models are not capable of. The next chapter will be the proof of concept, applying the new model to the scenarios discussed in Chapter 3.

Chapter 5

Proof of Concept

In this chapter, a prototype architecture of the system is offered based on the model proposed and the prototype architecture will be applied to the scenarios discussed above for proof of concept.

5.1 Closed Scenario

5.1.1 System Prototype Architecture Overview

For a closed scenario, the system consists of two parts, namely the server side component and the client side component. The server side stores all the patients' records, the list of devices, users who could access the system and their corresponding rights.

Besides that, each part contains an enforcement point performing access control. The enforcement point receives usage requests from the user and makes decisions.

With the eHealth system scenario, when a patient pays a visit to the doctor, first the doctor needs to log into the eHealth system by providing a valid username and password. The system checks the rights the doctor has like what records the doctor can get access to. Also the history-based engine will be queried to see if the doctor has a history of violating the policy. After successfully authorizing the doctor based on his/her role, the doctor can send a request to the system to get access to the particular patient's record. If the request gets approved, the system then releases a package including the record along with a UCON policy to the client side which the doctor is using.

The client side, once receiving the record, checks the integrity of the package and then extracts the patient's record and the UCON policy. While the doctor is accessing the record, the client system monitors the doctor's behavior and if a violation is found, for example, the doctor tries to copy the record using a flash drive when he/she is not supposed to do that, the system revokes the access rights and logs the event to the history-based engine. Also the system could report that violation to a third person. For hand-held devices like PDAs, the system also monitors the physical position of the device by using GPS or WIFI signal. If the device is out of the range of the hospital, the system will revoke the access rights to the record temporarily.

If the doctor needs to delegate his/her rights to another person, he/she sends the request to the server. Because it is the doctor's responsibility to check whether to grant the permission to the person, the server only needs to check if that person is within the system. If so, the person will be given the rights.

When an emergency occurs and the doctor needs to access the record for which he/she normally does not have the permission, called the break-glass policy here, the doctor sends a special request to the system. The system will then grant him/her the permission temporarily and also will log and report this special access session using a pre-defined method for later re-

evaluation.

After the treatment session ends, the client side system will keep monitoring the record for a specific period of time. During that time if the record is deleted by the doctor then the whole access session ends. Otherwise when reaching the maximum waiting time and the record is still in the device, the system will revoke the doctor's access rights and log this violation event to the history-based engine.

5.1.2 Obligation Policy Enforcement Point

Based on the previous discussion, the obligations here could be classified into two parts, namely the obligations that need a fulfillment check (here we call this kind of obligations *non-trusted obligations*) and obligations that do not need it (*trusted obligations* for short). These two kinds of obligations should be treated separately. Trust obligations are obligations taken by the system so this kind of obligations do not need the enforcement mechanism to make sure the fulfillment of it. As a result, the obligation policy enforcement point only focuses on the non-trusted obligations.

For non-trusted obligations, they could be divided into two kinds based on the time for fulfillment check. The first kind is the obligations that need to be checked instantly. For example asking the user to provide a valid username and password combination when trying to access the system. For this kind of obligation, the enforcement point needs to express the obligation using a sequence of actions. Taking the login example, what the enforcement point needs to do is:

1. Check if the username provided is in the right form, for example, the username meets

the length requirement and contains only permitted characters.

2. Check whether the username is in the system's valid username database.

3. Check the password input by the user is in the right form.

4. Check whether the username and password match.

Only if all the tests by the enforcement point pass, the system will treat the obligation as fulfilled.

On the other hand, the second kind of obligation is the one needing to be checked after a specific period of time. In our scenario the doctor has to delete the patient's record within 30 days after the treatment session ends. For this type of obligation, there is a timer embedded in the enforcement point. The timer will trigger an event requiring the enforcement point to check the fulfillment of the obligations.

The enforcement point is tied closely with the history-based engine. Any violation will trigger the enforcement point to log that event to the history based engine. The implementation of the enforcement point is based on an XACML enforcement engine. The policy specifications of the enforcement point using XACML schema will be introduced in the next section.

5.1.3 Policy Specifications

Based on the discussion above, a UCON policy specification is developed using the XACML [37] policy specification. XACML stands for *eXtensible Access Control Markup Language*, which

is a declarative access control policy language based on XML to describe how to interpret the policies.

In the policy specifications, ADF related rules are represented by the *policy set* in XACML while ODF related rules are represented by the *StateAction* element in XACML. The policy schema is shown below: (Adapted from [36])

```
<?xml version="1.0" encoding="UTF-8"?>
<schema>
        <complexType name="StateActionType">
                <sequence minOccurs="0">
                        <element ref="ucon:StateAction"></
                           element>
                </sequence>
        </complexType>
        <element name="DeniedPolicy" type="
           ucon:StateActionType"></element>
        <element name="ExitPolicy" type="ucon:StateActionType"
          ></element>
        <complexType name="StatePolicy">
                <sequence>
                        <element ref="ucon:StateAction"></
                           element>
                        <element ref="xacml:PolicySet"></
                           element>
                </sequence>
        </complexType>
        <element name="RequestcheckPolicy" type="
           ucon:StatePolicy"></element>
        <element name="OngoingcheckPolicy" type="
           ucon:StatePolicy"></element>
        <element name="EndedpostcheckPolicy" type="
           ucon:StatePolicy"></element>
        <element name="RevokedpostcheckPolicy" type="
           ucon:StatePolicy"></element>
        <complexType name="UCONPolicyType">
                <sequence>
                        <element ref="ucon:RequestcheckPolicy"
                           ></element>
                        <element ref="ucon:OngoingcheckPolicy"
                           ></element>
                        <element ref="
```

```
                    ucon:EndedpostcheckPolicy"></
                    element>
25                <element ref="
                    ucon:RevokedpostcheckPolicy"></
                    element>
26                <element ref="ucon:DeniedPolicy"></
                    element>
27                <element ref="ucon:ExitPolicy"></
                    element>
28            </sequence>
29            <attribute name="UCONPolicyId" type="anyURI"
                use="required"></attribute>
30        </complexType>
31        <element name="UCONPolicy" type="ucon:UCONPolicyType">
            </element>
32        <element name="StateAction" type="ucon:StateActions"><
            /element>
33        <complexType name="StateActions">
34            <sequence>
35                <element ref="xacml:Obligations"></
                    element>
36            </sequence>
37        </complexType>
38</schema>
```

The above schema shows all the elements of a UCON policy. In the schema, *UCONPolicy* represents the root element of the policy, which consists of all the rules. *RequestcheckPolicy*, *OngoingcheckPolicy*, *EndedpostcheckPolicy* and *RevokedpostcheckPolicy* are the type of *StatePolicy*, which consists of the *StateAction* element and the *PolicySet* of XACML. These are the ADF related rules. On the other hand, the *DeniedPolicy* and *ExitPolicy* have only the *StateAction* element because they are the ODF related rules.

A UCON policy specification can be easily developed based on this schema to configure the enforcement engine.

5.1.4 History-based Engine

As we discussed in last chapter, the history-based engine is used to store the past behaviors of the user and provide that information to the system when requested. An event in the event database is expressed using XML. For proof of concept use we have developed an XML schema for the event database as shown below:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="History">
<xs:compexType>
<xs:sequence>
        <xs:element name="Record">
        <xs:compexType>
        <xs:sequence>
                <xs:element name="UserID" type="xs:string"></
                    xs:element>
                <xs:element name="ResourceID" type="xs:string"
                    ></xs:element>
                <xs:element name="Violation" type="xs:string">
                    </xs:element>
                <xs:element name="TimeOfViolation">
                <xs:complexType>
                <xs:sequence>
                        <xs:element name="Date" type="xs:date"
                            ></xs:element>
                        <xs:element name="Time" type="xs:time"
                            ></xs:element>
                </xs:sequence>
                </xs:complexType>
                </xs:element>
        </xs:sequence>
        </xs:complexType>
        </xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
```

In the event database, four elements are stored for later query use: the user ID which could be used to identify the user involved in the particular event (*UserID*), the related resource

(*ResourceID*), the type of violation (*Violation*) which is expressed as a pre-defined number and a time stamp indicating when the event is taken place (*TimeOfViolation*).

The query and insertion action is done using the XQuery, which is an XML query language. In Appendix A we give an example of querying and insertion for an event database.

5.2 Open Scenario

5.2.1 System Prototype Architecture Overview

Because of the unique character of the open system, which is that the user of the system is not pre-defined, it is not possible to control the user-side devices. The system for the open system, here in our example the cloud computing system, only has one part which deploys on the server side.

When a user tries to get service from the cloud, he/she first needs to register on the system. If the service is not free of charge, the user also may need to pay for the service. The system checks the information the user inputs; also based on the information, the system queries the history-based engine to see if there exists a record of that user for evaluation. Then the system will show the user a term-of-service to inform the user his/her rights and obligations. After the user accepts the term-of-service, the system will grant him/her the corresponding rights in the system.

During the usage session, the system keeps monitoring the user's actions and updating any user attributes if needed. If abnormal access action is found, the system will notify the user, log the

event in the system and even revoke the access rights of the user.

After the usage session ends, the system will inform the user about his/her obligations, which may include giving feedback to the system, or paying for the service the user used if this has not been done before the usage session. The system will track the fulfillment of the obligations for a period of time and for those obligations which are not fulfilled, the system will log the event to the history-based engine along with reporting that to a specific third party.

5.2.2 Obligation Policy Enforcement Point

Similar to the policy enforcement point in a closed system, the one for an open system will also mainly focus on non-trusted obligations.

Because of the unique characteristics of the users for an open system, the users may violate some of the obligations because they are not familiar with the detailed requirements of the system and once this happens the system should have some mechanisms to give the users a chance to defend themselves. So based on this thought, in order to enhance the flexibility and user experience, a negotiation module is embedded in the enforcement point. Similar work has been done in [38].

Usually the negotiation process could be divided into three levels as shown in Figure 5.1 namely attributes query, attributes automatic negotiation and artificial negotiation. Attribute query will start querying user's attribute when attributes are insufficient, and negotiation will end once the wanted attributes are obtained. Otherwise automatic attribute negotiation will take effect. This negotiation step will help getting the wanted attributes according to attribute privacy policies of both sides, and negotiation will end if the wanted attributes are obtained. If

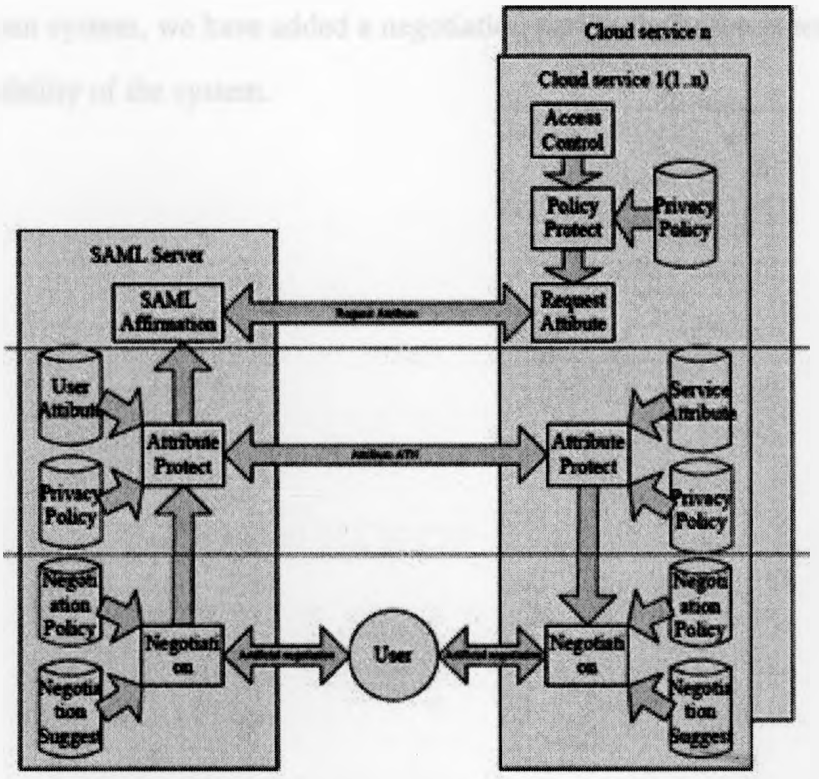

Figure 5.1: Negotiation Model (From [38])

both of the above levels fail, then the final level, artificial negotiation, will take effect. At this level, the system will ask a human user to make the decisions.

5.3 Summary

In this chapter, by developing a system prototype architecture based on our proposed model and applying the architecture of the prototype to our two scenarios for proof-of-concepts use, we proved the model fits well in our scenarios and solves the obligations problem there. Then a obligation policy specification used in enforcement point is developed using the XACML policy specification language.

Two enforcement point architectures are proposed based on the two scenarios. In the enforce-

ment point for open system, we have added a negotiation part to the enforcement point which enhances the flexibility of the system.

Chapter 6

Conclusions and Future Work

In this chapter, we present the conclusion of the thesis. We begin this chapter with the summary of our work and contributions in this thesis. Then we discuss some possible future research directions

6.1 Conclusions

This thesis is mainly focused on the obligation part of the usage control model. After a detailed review of previous literature, based on the previous model, a new one is proposed to handle the problem the current ones cannot. The main work and contributions of this thesis are summarized below:

1. We proposed a new obligation model for UCON. There are several improvements in this new model:

- We added post-check process in the state transition scheme to fit the post-obligation requirement better.

- For the enforcement mechanism, the decision function is put into the OS level to make the enforcement more secure while at the same time, other parts of the enforcement model remain in the application level for flexibility and bridging the gap between the OS level and the application.

- We add a history-based policy engine to the enforcement model to log the history of access session information for later evaluation use.

2. We offered two application scenarios for proof-of-concepts purposes, one eHealth scenario as the one for closed system and one cloud computing scenario as the one for open system.

3. We proposed a prototype architecture of the system based on the proposed model, developed the UCON policy specification using XACML schema, added a negotiation model to the enforcement point for the open system.

4. We proposed a Proof-of-Concept by applying the prototype architecture to the two scenarios and showed the model proposed fits in the scenarios well.

6.2 Future Work

Usage control is a relatively new research area in the access control field so there remains lots of work to be done in the future. As for the work in this thesis, first we only proposed

a prototype architecture for proof-of-concept here. For a more accurate proof, the next step maybe is to implement the prototype in order to test it in a real world application.

Second, the history-based engine could be studied further. For example, what kind of information should be passed to the engine. In our example, only the *UserID*, *ResourceID*, *TimeOfViolation* and *ViolationType* of one violation event is stored in the history-based engine. The next step we should consider is whether this is enough. Also, after the event database gives back results for a query, how this information should be used is another issue-should all the records be treated equally or is there a weight to each entry.

Bibliography

[1] Department of Defense National Computer Security Center: *Department of Defense Trusted Computer Systems Evaluation Criteria*, December 1985. DoD 5200.28-STD

[2] Ferraiolo, D.F. and Kuhn, D.R.: *Role-Based Access Control*, 15th National Computer Security Conference, pp. 554-563, Elsevier Advanced Technology Publications, October 1992

[3] Jaehong Park, Ravi Sandhu: *The $UCON_{ABC}$ Usage Control Model*, ACM Transactions on Information and System Security, Vol. 7, No. 1, pp. 128-174, Feb. 2004

[4] American National Standards Institute: *For information technology - role-based access control*, ANSI INCITS 359, Jan 2004

[5] Matunda Nyanchama, Sylvia L. Osborn: *Access Rights Administration in Role-Based Security Systems*, DBSec 1994, pp. 37-56, Aug. 1994

[6] Sylvia L. Osborn: *Role-based access control*, Milan Petkovic and Willem Jonker, editors, Security, Privacy and Trust in Modern Data Management. Springer, 2007

[7] Jaehong Park, Ravi Sandhu: *Towards Usage Control Models: Beyond Traditional Access Control*, Proceedings of the seventh ACM symposium on Access control models and technologies, SACMAT 2002. pp. 57-64, June,2002

[8] Aliaksandr Lazouski, Fabio Martinelli, Paolo Mori: *Usage control in computer security: A survey*, Computer Science Review (2010), pp. 81-99, doi:10.1016/j.cosrev.2010.02.002

[9] Q. Liu, R. Safavi-Naini, N.P. Sheppard: *Digital rights management for content distribution*, ACSW Frontiers'03: Proceedings of the Australasian Information SecurityWorkshop, Australian Computer Society, Inc., Darlinghurst, Australia, pp. 49-58, 2003

[10] W. Ku, C.-H. Chi: *Survey on the technological aspects of digital rights management*, Information Security, pp. 391-403, 2004

[11] M. Xu, X. Jiang, Ravi Sandhu, X. Zhang: *Towards a VMM-based usage control framework for OS kernel integrity protection*, SACMAT'07: Proceedings of the 12th ACM Symposium on Access Control Models and Technologies, ACM, NewYork, NY, USA, pp. 71-80, 2007

[12] D. Kyle, J.C. Brustoloni: *Uclinux: A linux security module for trusted-computing-based usage controls enforcement*, STC'07: Proceedings of ACM Workshop on Scalable Trusted Computing, ACM, New York, NY, USA, pp. 63-70, 2007

[13] M. Alam, J.-P. Seifert, Q. Li, X. Zhang: *Usage control platformization via trustworthy SELinux*, ASIACCS'08: Proceedings of ACM Symposium on Information, Computer and Communications Security, ACM, New York, NY, USA, pp. 245-248, 2008

[14] Fabio Martinelli, Paolo Mori: *A model for Usage Control in GRID Systems*, Security and Privacy in Communications Networks and the Workshops, pp 169-175, 2007

[15] I. Foster: *Globus toolkit version 4: Software for service-oriented systems*, Proceedings of IFIP International Conference on Network and Parallel Computing, volumn 3779, page 2-13. Springer-Verlag, LNCS, 2005

[16] A. Berthold, M. Alam, R. Breu, M. Hafner, A. Pretschner, J.-P. Seifert, X. Zhang: *A technical architecture for enforcing usage control requirements in service-oriented architectures*, SWS'07: Proceedings of ACM Workshop on Secure Web Services, ACM, New York, NY, USA, pp. 18-25, 2007

[17] Alexander Pretschner, Fabio Massacci, Manuel Hilty: *Usage Control in Service-Oriented Architectures*, C.Lambrinoudakis, G. Pernul, A M. Tjoa (Eds.): TrustBus 2007, LNCS 4657, pp. 83-93. 2007

[18] M. Hilty, A. Pretschner, C. Schaefer, T. Walter: *Usage control requirements in mobile and ubiquitous computing applications*, ICSNC'06: Proceedings of the International Conference on Systems and Networks Communication, IEEE Computer Society, Washington, DC, USA, pp. 27-32, 2006.

[19] Andreas Schaad, Jonathan D. Moffett: *Delegation of Obligation*, Proceedings of the Third International Workshop on Policies for Distributed Systems and Networks (POLICY'02). pp. 25-35, 2002

[20] Andreas Schaad: *Revocation of Obligation and Authorization Policy Objects*, S. Jajodia and D. Wijesekera (Eds.): Data and Applications Security 2005, LNCS 3654, pp. 28-39,

2005

[21] W. H. Winsborough, K. E. Seamons, V. E. Jones: *Automated trust negotiation*, DARPA Information Survivability Conference and Exposition, pp. 88-102, Jan. 2000

[22] Lars Olson, Marianne Winslett, Gianluca Tonti, Nathan Seeley, Andrzej Uszok, Jeffrey Bradshaw: *Trust Negotiation as an Authorization Service for Web Services*, Proceedings of the 22nd International Conference on Data Engineering Workshops (ICDEW'06), pp.21-30, 2006

[23] Adam J. Lee, Marianne Winslett, Jim Basney, Von Welch: *Traust: A Trust Negotiation Based Authorization Service*, iTrust 2006, LNCS 3986. pp. 458-462. 2006

[24] M. Winslett, T. Yu, K. E. Seamons, A. Hess, J. Jacobson, R. Jarvis, B. Smith, L. Yu: *The TrustBuilder architecture for trust negotiation*, IEEE Internet Computing, 6(6):30-37, Nov./Dec. 2002.

[25] http://en.wikipedia.org/wiki/System, Dec 2010

[26] Khaled Gaaloul, Francois Charoy, Andreas Schaad: *Modelling task delegation for human-centric eGovernment workflows*, Proceedings of the 10th Annual International Conference on Digital Government Research: Social Networks: Making Connections between Citizens, Data and Government (dg.o '09), Digital Government Society of North America 79-87.

[27] R4eGov Technical Annex 1: *Towards e-Administration in the large*, Sixth Framework Programme, Information Society Technologies, March 2006

[28] Vincenzo Della Mea: *What is e-Health (2): The death of telemedicine?*, J Med Internet Res 2001;3(2):e22

[29] http://aws.amazon.com/ec2/, Dec 2010

[30] Achim D. Brucker Helmut Petritsch: *Extending access control models with break-glass*, Proceedings of the 14th ACM symposium on Access control models and technologies (SACMAT '09). ACM, New York, NY, USA, pp. 197-206, 2009

[31] Gerald Vogt: *Multiple authorization - a model and architecture for increased, practical security*, Proceedings of IFIP/IEEE Symposium on Integrated Network Management, pp. 109-112, 2003

[32] Foster, I.,Zhao, Y.: *Cloud Computing and Grid Computing 360-Degree Compared*, Grid Computing Environments Workshop (2008), pp. 1-10, 2008

[33] http://en.wikipedia.org/wiki/Cloud_computing, Dec 2010

[34] Ravi Sandhu, Jaehong Park: *Usage Control: A vision for next generation access control*, Computer Network Security, Second International Workshop on Mathematical Methods, Models, and Architectures for Computer Network Security, MMM-ACNS 2003, St. Petersburg, Russia, September 21-23, 2003

[35] X. Zhang, F. Parisi-Presicce, Ravi Sandhu J. Part: *Formal model and policy specification of usage control*, ACM Transactions on Information and System Security, 8(4):351-387, 2005

[36] Basel Katt, Xinwen Zhang, Ruth Breu, Michael Hafner, Jean-Pierre Seifert: *A general obligation model and continuity: enhanced policy enforcement engine for usage control*, Proceedings of the 13th ACM symposium on Access control models and technologies (SACMAT '08). ACM, New York, NY, USA, pp.123-132, 2008

[37] eXtensible Access Control Markup Language (XACML) Version 3.0, Committee Specification 01, AUG, 10, 2010. http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-cs-01-en.pdf, Dec 2010

[38] Chen Danwei, Huang Xiuli, Ren Xunyi: *Access Control of Cloud Service Based on UCON*, CloudCom 2009, LNCS 5931, pp. 559-564, 2009

[39] http://exist.sourceforge.net/, Jan 2011

Appendix A

Example Testing of Event Database

In this appendix, we give some examples using the XQuery language to query an example event database and insert new data to the database. An example database based on the schema provided in Chapter 5 is listed below and the examples are based on it. All examples here run on the eXist-db [39], which is an open source native XML database.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<History>
        <Record>
                <UserID>U001</UserID>
                <ResourceID>R100</ResourceID>
                <Violation>01</Violation>
                <TimeOfViolation>
                        <Date>2011-01-11</Date>
                        <Time>09:00:00</Time>
                </TimeOfViolation>
        </Record>
        <Record>
                <UserID>U002</UserID>
                <ResourceID>R355</ResourceID>
                <Violation>01</Violation>
                <TimeOfViolation>
                        <Date>2010-02-04</Date>
                        <Time>17:15:23</Time>
                </TimeOfViolation>
```

```
20          </Record>
21          <Record>
22                  <UserID>U001</UserID>
23                  <ResourceID>R367</ResourceID>
24                  <Violation>04</Violation>
25                  <TimeOfViolation>
26                          <Date>2010-06-15</Date>
27                          <Time>01:02:23</Time>
28                  </TimeOfViolation>
29          </Record>
30          <Record>
31                  <UserID>U002</UserID>
32                  <ResourceID>R276</ResourceID>
33                  <Violation>03</Violation>
34                  <TimeOfViolation>
35                          <Date>2010-12-22</Date>
36                          <Time>13:32:16</Time>
37                  </TimeOfViolation>
38          </Record>
39          <Record>
40                  <UserID>U005</UserID>
41                  <ResourceID>R847</ResourceID>
42                  <Violation>02</Violation>
43                  <TimeOfViolation>
44                          <Date>2010-01-22</Date>
45                          <Time>06:28:04</Time>
46                  </TimeOfViolation>
47          </Record>
48          <Record>
49                  <UserID>U008</UserID>
50                  <ResourceID>R639</ResourceID>
51                  <Violation>04</Violation>
52                  <TimeOfViolation>
53                          <Date>2010-11-03</Date>
54                          <Time>23:13:11</Time>
55                  </TimeOfViolation>
56          </Record>
57  </History>
```

The most common action for query is to query by the user ID, for example if we want to query all information by userID=U002, the following command could be applied:

```
//Record[UserID='U002']
```

When executed, the result given back is

```
1
<Record>
        <UserID>U002</UserID>
        <ResourceID>R355</ResourceID>
        <Violation>01</Violation>
        <TimeOfViolation>
                <Date>2010-02-04</Date>
                <Time>17:15:23</Time>
        </TimeOfViolation>
</Record>
2
<Record>
        <UserID>U002</UserID>
        <ResourceID>R276</ResourceID>
        <Violation>03</Violation>
        <TimeOfViolation>
                <Date>2010-12-22</Date>
                <Time>13:32:16</Time>
        </TimeOfViolation>
</Record>
```

Next we will give an example of the command for insertion:

```
update insert
<Record>
        <UserID>U002</UserID>
        <ResourceID>R111</ResourceID>
        <Violation>02</Violation>
        <TimeOfViolation>
                <Date>2011-01-15</Date>
                <Time>19:23:08</Time>
        </TimeOfViolation>
</Record>
into //History
```

After executing, the contents of the event database is shown below:

```
<History>
        <Record>
                <UserID>U001</UserID>
                <ResourceID>R100</ResourceID>
                <Violation>01</Violation>
```

```
6                <TimeOfViolation>
7                        <Date>2011-01-11</Date>
8                        <Time>09:00:00</Time>
9                </TimeOfViolation>
10       </Record>
11       <Record>
12               <UserID>U002</UserID>
13               <ResourceID>R355</ResourceID>
14               <Violation>01</Violation>
15               <TimeOfViolation>
16                       <Date>2010-02-04</Date>
17                       <Time>17:15:23</Time>
18               </TimeOfViolation>
19       </Record>
20       <Record>
21               <UserID>U001</UserID>
22               <ResourceID>R367</ResourceID>
23               <Violation>04</Violation>
24               <TimeOfViolation>
25                       <Date>2010-06-15</Date>
26                       <Time>01:02:23</Time>
27               </TimeOfViolation>
28       </Record>
29       <Record>
30               <UserID>U002</UserID>
31               <ResourceID>R276</ResourceID>
32               <Violation>03</Violation>
33               <TimeOfViolation>
34                       <Date>2010-12-22</Date>
35                       <Time>13:32:16</Time>
36               </TimeOfViolation>
37       </Record>
38       <Record>
39               <UserID>U005</UserID>
40               <ResourceID>R847</ResourceID>
41               <Violation>02</Violation>
42               <TimeOfViolation>
43                       <Date>2010-01-22</Date>
44                       <Time>06:28:04</Time>
45               </TimeOfViolation>
46       </Record>
47       <Record>
48               <UserID>U008</UserID>
49               <ResourceID>R639</ResourceID>
50               <Violation>04</Violation>
```

```
51                  <TimeOfViolation>
52                          <Date>2010-11-03</Date>
53                          <Time>23:13:11</Time>
54                  </TimeOfViolation>
55          </Record>
56          <Record>
57                  <UserID>U002</UserID>
58                  <ResourceID>R111</ResourceID>
59                  <Violation>02</Violation>
60                  <TimeOfViolation>
61                          <Date>2011-01-15</Date>
62                          <Time>19:23:08</Time>
63                  </TimeOfViolation>
64          </Record>
65  </History>
```