



PART 3:

**CHAPTER 7:
THE OAC(UCON) MODEL**

**CHAPTER 8:
THE PROTOTYPING AND MODEL EVALUATION**

**CHAPTER 9:
CONCLUSION**

CHAPTER 7:

THE OAC(UCON) MODEL

7.1 Introduction

This chapter introduces the concept of the **Optimistic Access Control with Usage Control** model, designated the OAC(UCON) model. In the previous chapters the inadequacies of traditional access controls models were highlighted, and it became clear that the requirements for the OAC(UCON) had to include flexibility, adaptability and an open architecture. However, there should also be provisos that prevent abuse of the openness offered by this model. An example of the model that meets these requirements is presented at this juncture.

7.2 A motivating example

Suppose company ABC is an e-Recruitment company where clients and prospective candidates (job seekers) place job orders and applications respectively online. Company ABC then maintains databases of candidates and clients. While internally the company places access controls on sensitive information such as salaries, the information for collaborative job matching is unrestricted. Suppose an employee decides to download all the telephone numbers that are available on these databases and sells it to a telemarketer. Due to the lack of deterrents, this act is relatively easy to carry out. Furthermore, the employee may claim that he was unaware of the fact that his act was unethical. This type of security breach is typically blamed on the user and a lack of user training. The rationale behind such a security breach is contradictory to the security usability requirements that should be implemented in a software system. According to Zurko (2005) we have to ask, 'why did the system make the insecure option so easy and attractive' – and in this case ultimately lucrative? Perhaps if

synchronous system deterrents had been deployed at the time of usage, the employee may have been deterred from carrying out an illegitimate act.

In this scenario, the following stipulations or mechanisms could have been used as usage control deterrents:

- **Pre-obligation:** The user must click on a button in a window to indicate that he/she agrees not distribute this information.
- **Ongoing obligation:** A window with the following warning '*This dataset must be used exclusively for work-related purposes ONLY*' is to remain open at all times.
- **Pre-condition:** The user is warned beforehand that 'this information may be accessed during business hours only'.
- **Ongoing conditions:** The information may be accessed during business hours only (same as the pre-condition as it is time dependent). However, although the user may have accessed the information during business hours (i.e. initially satisfying all the conditions), the ongoing condition may become invalid as time passes.
- **Post-obligation:** The user must indicate the priority of the task completed if he/she actually accessed these databases outside business hours. This information could be used to develop a profile on the user.

The post-obligation implies that an employee may in fact access the databases after hours. Under the optimistic paradigm the employee should ultimately be able to download the data in the case of an emergency. This is permitted, as the employee should not be hampered in the performance of his/her duties. While fulfilling the post-obligations facet is within a user's control, the pre-conditions and ongoing conditions are not. For this reason a break-the-glass tool may be included to allow for overriding the pre-conditions or ongoing conditions.

The Break-the-Glass policy

The Break-The-Glass (BTG) policy provides a mechanism to override access control policies (Ferreira et al., 2006) as part of the access control policy stated in the previous section. This act can be justified because there are situations when access is required, even if it means

that confidentiality is breached. The important issue is that this breach is openly declared to the responsible parties and the access is properly analysed afterwards. At that stage it can then be considered whether the breach was well justified or whether it was an intrusion.

The following section investigates how such deterrents may be practically implemented under the optimistic paradigm.

7.3 Architecture

Many systems are based on a three-tiered architecture – access is via the web, the application programs reside within an application server, and the data is stored within a database system (Li et al., 2005). Only the application tier is considered in the next section (Figure 7-1).

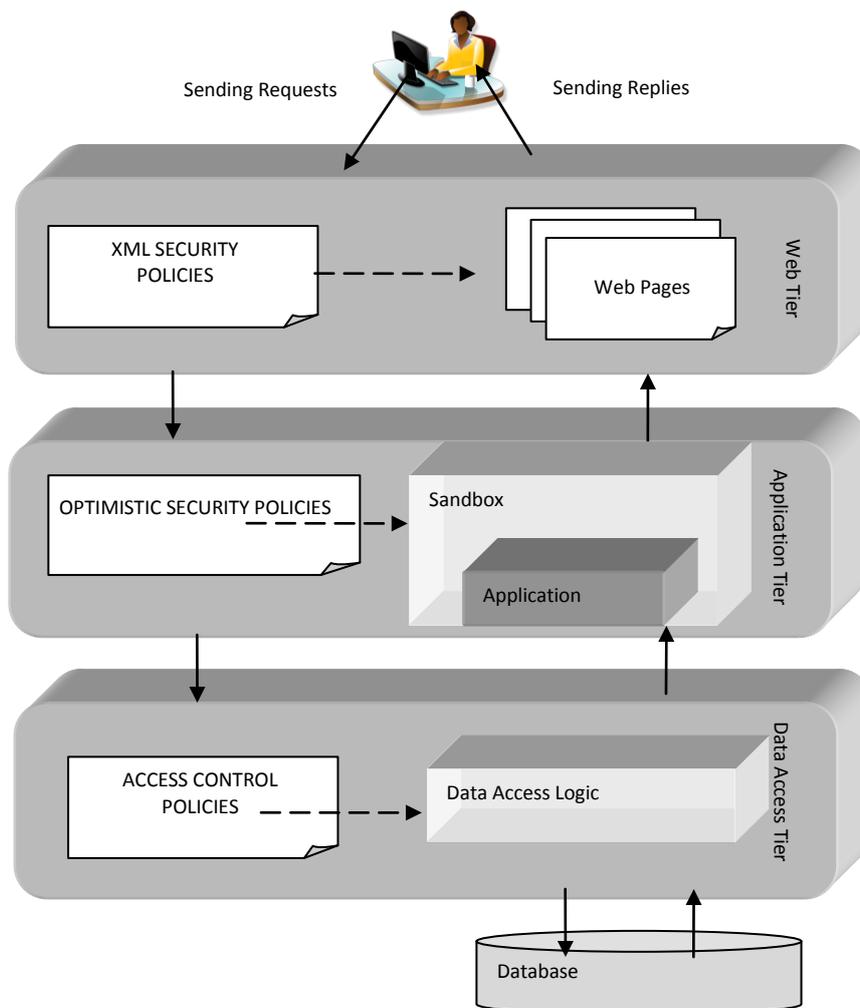


Figure 7-1: Architectural Diagram

In Chapter 8 the prototype is designed within the Application Layer where it is protected by the Sandbox offered by the Java Authentication and Authorisation Service (JAAS). In the basic Java security model, trusted code is allowed full access to the system, and untrusted code is forced to execute in a restricted environment called the sandbox. The access control policies of the sandbox are established by an instance of the `SecurityManager` class. A security manager provides methods that determine whether a particular operation is permissible (Tymann and Schneider, 2008).

The conceptual structure of the OAC(UCON) model is shown in Figure 7-2. The **Usage Enforcement Facility** includes the Customisation Module, which can be used to constrain a user's access to specific components of an object. Within the model, it is presumed that as the user's optimistic rights are downgraded, the data that he/she is allowed to access may be constrained according to sensitivity levels. The Monitoring Module involves logging all accesses, while the Update Module is involved in changing the access rights.

The **Usage Decision Facility** includes the Condition, Obligation, Break-the-Glass and Authorisation Modules. In general, most accesses are allowed – except for those users who breached the system in the past. The Authorisation Module therefore determines the level of access that is permitted. For example, an authenticated user is permitted to view data of all sensitivity levels. In most cases users are trusted to view all objects that are given optimistic access rights. The Condition Module decides whether conditional requirements for authorised requests are satisfied or not by using usage rules and contextual information (such as current time, IP address, etc.). The Obligation Module decides whether certain obligations have to be performed or not, either, before, during or after the requested usage has been performed. If there exists any post-obligation that has to be performed, this must be monitored by the Monitoring Module and the result has to be resolved by the Update Module in the Usage Decision Facility. If the user does not satisfy the obligations before or during the access, he/she is not permitted to access the information. However, when the conditions are invalid, the user is allowed to access the information by using the Break-the-Glass Module. This Module is intended for emergencies only and the user is clearly informed

about the consequences of accessing this information illegitimately. Access by ‘breaking the glass’ is always red flagged for auditing purposes.

The **Usage Protection Facility** is used to protect the integrity of the information. The Audit Module will check through all accesses and identify possible illegal accesses. It will additionally create a list of red-flagged accesses. If the user cannot justify an illegal access, then the Authorisation Module will restrict the user's optimistic access rights in future. This may also involve punitive action. If the user has performed some illegal modification to the data, the Roll-back Module will attempt to return the data back to its original state.

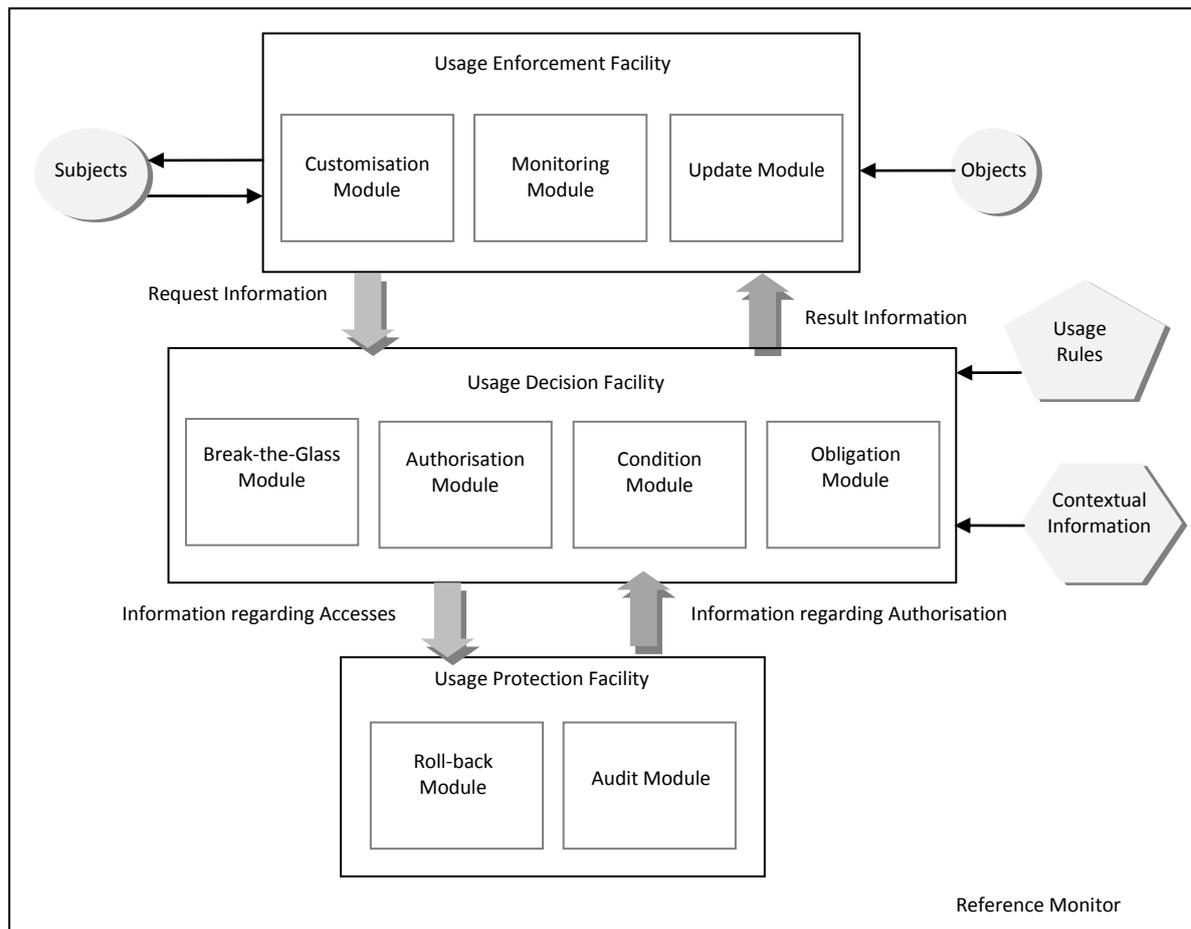


Figure 7-2: Conceptual Structure for Optimistic Access Control enhanced with Usage Control

The Break-the-Glass Module, which provides a mechanism to override access control policies, is used in the same way as by Ferrieria et al. (2006). It is important to note that the responsible parties are fully aware of this security breach and that the access is properly analysed afterwards so as to consider whether the breach was well justified or a mere intrusion. What is different from the way in which the Break-the-Glass policy is used by Ferrieria et al. (2006) is that with the OAC(UCON) Model it is enforced when the system conditions are not satisfied. For instance: users are only allowed to access data from 8:00am to 5:00pm. In an emergency, the user will be allowed to override this system condition by making use of the Break-the-Glass facility.

The sequence of a user's interaction with the OAC(UCON) model can be summarised as follows:

1. The Authorisation Module checks the constraints on the information requested by using the Customisation Module. In general, the user is allowed to access the information if he/she is authenticated and the data is under the optimistic access control domain.
2. Before the user is allowed access, he/she has to fulfil one or more specific pre-obligations set up by the Obligation Module. If these obligations are not met, the user is not allowed to access the information.
3. If the system pre-conditions are valid, the user is allowed to access the information. These permissions are maintained by the Condition Module. However, if the system conditions are not valid, the user is presented with an opportunity to use the Break-the-Glass facility deployed by the Break-the-Glass Module. The user is given adequate warning about the usage of this facility. Moreover, it is red-flagged and logged.
4. The system is frozen until the user decides whether or not to use the Break-the-Glass facility.
5. If the user employs the Break-the-Glass facility, the system verifies whom it needs to notify and proceeds with the access.
6. All notifications and user actions are registered automatically by the Monitoring Module.

7. During the access, the user has to meet the ongoing obligations. If they are not met, the user's access to the information is automatically revoked.
8. During the access, the ongoing conditions may become invalid. The user then has the opportunity to use the Break-the-Glass facility to continue with the access.
9. After the access, the user may have to satisfy a post-obligation. This will be monitored by the Monitoring Module.
10. The Update Model may change the access rights of a user if he/she has committed an unjustified breach.
11. The Audit Model checks for red flags and unjustified breaches.
12. The Roll-back Module may be deployed if an unjustified breach resulted in the data being compromised.

The OAC(UCON) model is intended to fit into a mixed-initiative access control framework (see Figure 7-3), encompassing traditional access control such as role-based access control for highly classified information and optimistic access control enforced with usage control for information that is unclassified. Under the optimistic access control paradigm, users should be allowed access under most circumstances. Thus, access would not depend on the subject attributes or the object attributes. It is assumed that data is freely available and not subject to authorisation unless the user's optimistic rights have been downgraded. While the user will not be permitted to access the information unless the obligations are satisfied, he/she will under special circumstances be allowed to access the data by utilising the Break-the-Glass facility – even if the pre-condition(s) or ongoing condition(s) are invalid.

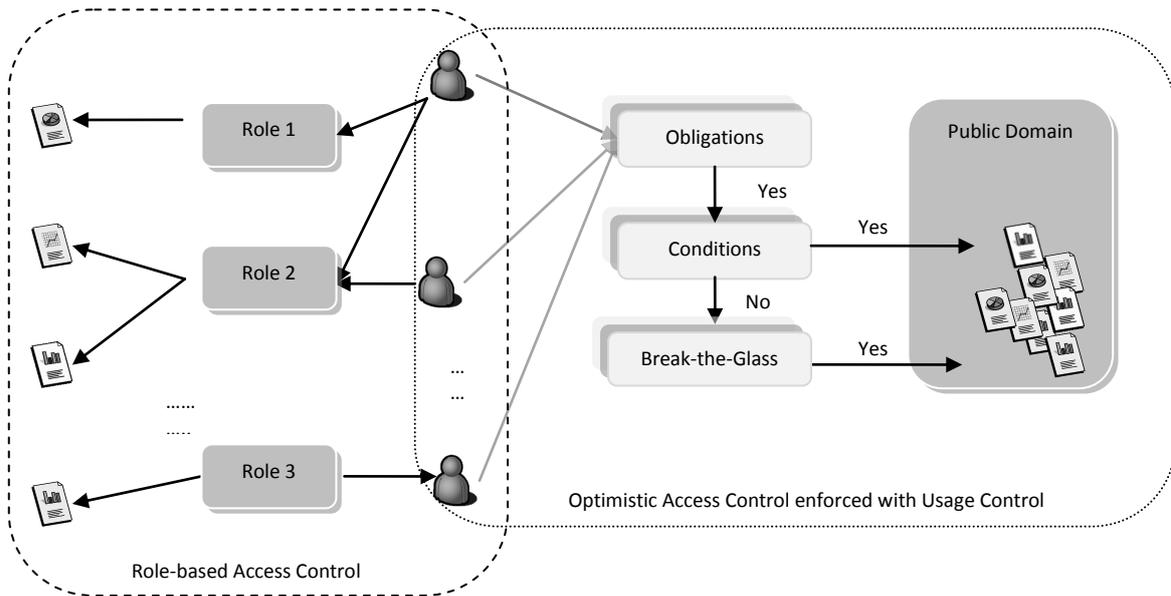


Figure 7-3: A Mixed-Initiative Access Control Framework – combining RBAC with OAC(UCON)

7.4 Detailed Design

7.4.1 Formal Specifications

The formalised definition of the optimistic access control enhanced with usage control is as follows:

Pre-Authorisation:

- Subjects S , Objects O , Rights R
- Function that checks if the user s , requesting access o with right r has Optimistic Rights:
 $\text{OptimisticRights}(s,o,r)$
- Function that checks if the set of Pre-Conditions are satisfied or not:
 $\text{PreC}(s,o,r)$
- Function that checks if the set of Pre-Obligations are satisfied or not:
 $\text{PreB}(s,o,r)$

- Function that requests if the user wants to override pre-conditions or not by using the Break-the-Glass facility:

Break-the-Glass(s,o,r)

- Access is allowed momentarily if the pre-obligations and pre-conditions are satisfied:

$$\begin{aligned} \text{allowed}(s,o,r) \Rightarrow & \text{OptimisticRights}(s,o,r) && \rightarrow \{\text{true}, \text{false}\} \\ & \wedge \text{PreB}(s,o,r) && \rightarrow \{\text{true}, \text{false}\} \\ & \wedge \text{PreC}(s,o,r) && \rightarrow \{\text{true}, \text{false}\} \end{aligned}$$

- Otherwise allow the user to use the Break-the-Glass facility

$$\begin{aligned} \text{allowed}(s,o,r) \Rightarrow & \text{OptimisticRights}(s,o,r) && \rightarrow \{\text{true}, \text{false}\} \\ & \wedge \text{PreB}(s,o,r) && \rightarrow \{\text{true}, \text{false}\} \\ & \wedge \neg \text{PreC}(s,o,r) && \rightarrow \{\text{true}, \text{false}\} \\ & \wedge \text{Break-the-Glass}(s,o,r) && \rightarrow \{\text{true}, \text{false}\} \end{aligned}$$

- Access is revoked if the pre-obligations are not met or if the Break-the-Glass is not used when the pre-conditions are invalid:

$$\begin{aligned} \text{revokeAccess}(s,o,r) \Rightarrow & \neg \text{Break-the-Glass}(s,o,r) && \rightarrow \{\text{true}, \text{false}\} \\ & \vee \neg \text{PreB}(s,o,r) && \rightarrow \{\text{true}, \text{false}\} \end{aligned}$$

Ongoing Authorisations

- Subjects S , Objects O , Rights R
- Function that checks if the set of ongoing Conditions are satisfied or not:

onC(s,o,r)

- Function that checks if the set of ongoing Obligations are satisfied or not:

onB(s,o,r)

- Function that requests the user to override the ongoing conditions or not, using the Break-the-Glass facility:

Break-the-Glass(s,o,r)

- Sustain access if the ongoing obligations and ongoing conditions are momentarily satisfied:

$$\begin{aligned} \text{allowed}(s,o,r) \Rightarrow & \text{onB}(s,o,r) && \rightarrow \{\text{true}, \text{false}\} \\ & \wedge \text{onC}(s,o,r) && \rightarrow \{\text{true}, \text{false}\} \end{aligned}$$

- Otherwise allow the user to employ the Break-the-Glass facility if the ongoing conditions are invalid in order to sustain access:

$$\begin{aligned}
 \text{allowed}(s,o,r) \Rightarrow & \quad \text{onB}(s,o,r) & \rightarrow & \quad \{\text{true}, \text{false}\} \\
 & \quad \wedge & & \\
 & \quad \text{-onC}(s,o,r) & \rightarrow & \quad \{\text{true}, \text{false}\} \\
 & \quad \wedge & & \\
 & \quad \text{Break-the-Glass}(s,o,r) & \rightarrow & \quad \{\text{true}, \text{false}\}
 \end{aligned}$$

- Access is revoked if the ongoing obligations are not met or if the Break-the-Glass facility is not accepted when the ongoing conditions are invalid:

$$\begin{aligned}
 \text{-revokeAccess}(s,o,r) \Rightarrow & \quad \text{-onB}(s,o,r) & \rightarrow & \quad \{\text{true}, \text{false}\} \\
 & \quad \vee & & \\
 & \quad \text{-Break-the-Glass}(s,o,r) & \rightarrow & \quad \{\text{true}, \text{false}\}
 \end{aligned}$$

Post Authorisation

If a user does not satisfy their post-obligations or is involved in an unjustified breach, then the user's optimistic rights are downgraded and his/her access will be constrained in future.

PostUpdate (OptimisticRights (s,o,r)): OptimisticRights (s,o,r) \Rightarrow

$$\begin{aligned}
 & \quad \text{-PostObligations}(s,o,r) & \rightarrow & \quad \{\text{true}, \text{false}\} \\
 & \quad \vee & & \\
 & \quad \text{-JustifiedBreach}(s,o,r) & \rightarrow & \quad \{\text{true}, \text{false}\}
 \end{aligned}$$

Administrator-controlled attributes can be modified by administrative actions. These attributes are modified at the administrator's discretion but are 'immutable' in that the system does not modify them automatically. Mutable attributes are automatically modified by the system (Park et al., 2004). For instance, if the users cannot justify having used the Break-the-Glass procedure, their optimistic access rights may be revoked or constrained in future.

7.4.2 The Use Case Diagram of Usage Control under the Optimistic Access Control Paradigm

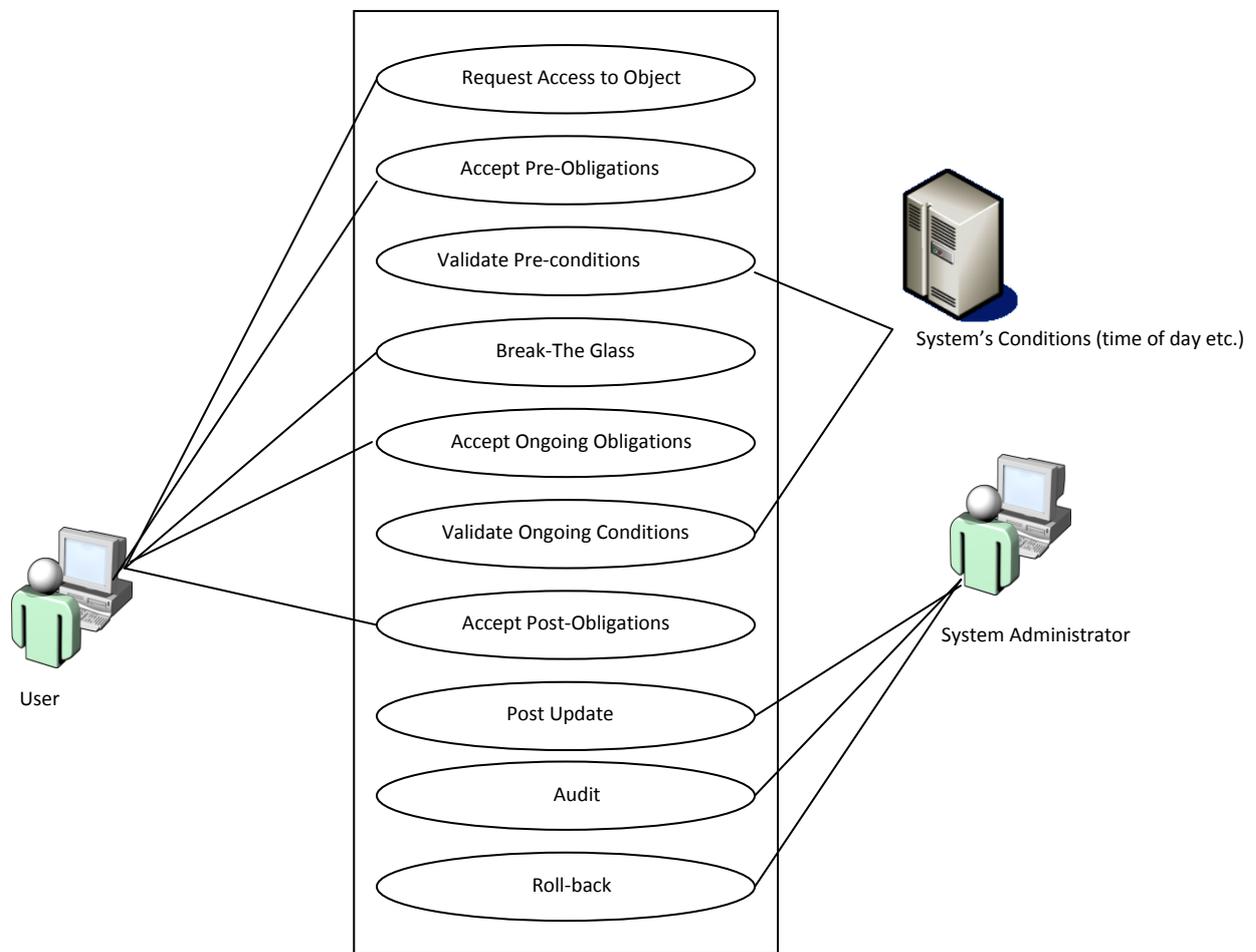


Figure 7-4: Use Case Diagram of OAC(UCON)

The implementation of pre-authorization is relatively simple as it warrants checking the conditions and obligations before the user may proceed. The implementation of ongoing authorisation is, however, non-trivial. The implementation decision taken to deal with this issue is to be addressed in Chapter 8. As the OAC(UCON) model is based on optimistic access control, most users are trusted to access information that is relevant to their context. Note that this is a less prescriptive approach than, say, role-based or mandatory access control. Due to the openness of the architecture, the user has to meet all pre-obligations and ongoing obligations, as he/she is expected to behave in a trustworthy manner.

In terms of the enforcement of security policies, it is imperative that this function be located centrally and enforced uniformly. The same notion would apply to the implementation of such policies in terms of application logic (Verhanneman et al., 2005). This type of deployment may be achieved through the use of aspect-oriented methodologies. The premise of the model is to create an aspect that will intercept calls when a subject requests access to an object and to enforce optimistic access control enhanced with usage control. A significant amount of work has been conducted in aspect-oriented security in respect of access control. The implementation of access control using aspect-oriented programming has been shown to ease the development of security-type concerns such as access control (De Win et al., 2001), as it results in an implementation that is easier to maintain and port to different environments.

According to Jones and Rastogi (2004), security controls may fall in one of four categories: *corrective control*, *deterrent control*, *detective control* and *preventative control*. Access control falls in the preventative control category. Information under this protection is typically secured in terms of roles or attributes. However, information under the public domain is not. Securing a distributed computing environment against malicious or otherwise disruptive use involves two aspects (Georgiev and Georgiev, 2001):

- Social, where the safeguarding of a computer system relies on social deterrents, such as shameful exposure or prosecution.
- Technical, where the system is protected by technical means, such as encryption algorithms and access controls.

Detective functions attempt to identify unwanted events while they are occurring or after they have occurred. Recovery controls restore lost computing resources or capabilities and help the organisation to recover monetary losses caused by a security violation. Corrective controls either remedy the circumstances that allowed the unauthorised activity or return conditions to what they were before the violation (Kim and Leem, 2004). Deterrent controls are intended to discourage individuals from intentionally violating information security policies or procedures. Typically, organisations implement deterrents such as anti-virus systems, passwords or they foster security awareness. However, with the use of the OAC(UCON) model, deterrents are achieved in a proactive manner.

7.5 Conclusion

This chapter presented the OAC(UCON) model together with specifications for a practical and implementable system. One of the criticisms levelled at the model in earlier presentations was its applicability. This problem was addressed by viewing the model within a mixed-initiative context, that is, within the context of traditional access controls. The model is highly applicable in contexts where some data cannot be reasonably protected by traditional access controls and needs to be openly available. This data can now be relegated to the public domain but remain subject to usage control rules under the optimistic access control paradigm. In the next chapter, the prototype will demonstrate a subset of the functionality of the OAC(UCON) that focuses more on the usage decisions component of the model. The prototype is implemented by using aspect orientation to demonstrate the suitability of the paradigm for this context. In addition, the model concept is evaluated in terms of the design science research methodology to assess its effectiveness and scalability.

CHAPTER 8:

PROTOTYPING AND MODEL EVALUATION

8.1 Introduction

The literature survey presented in Chapters 2, 3 and 4 has led to the development of a Usage Control Model for Optimistic Access Control (OAC(UCON) model) (Chapter 7). The current chapter presents the design of a proof-of-concept prototype to demonstrate a subset of the model concept, based specifically on the usage decision. The prototype was developed using an aspect-oriented programming language namely AspectJ. In addition, the model was evaluated in terms of the design science research method (March and Smith, 1995) to test its scalability and efficacy as a security measure. During this process, several evaluative prototypes were developed so as to verify the model concept for commercial systems. Hence, the evaluative prototype encompassed a larger subset of the model concept than the proof-of-concept prototype. The programming paradigm for the evaluative prototypes was not stipulated. Consequently, the evaluative prototypes were implemented by using various design techniques and programming languages in order to provide a more comprehensive assessment of the model.

8.2 The aim of the proof-of-concept prototype

The aim of the prototype was to provide key working points to show how the model can scale up using aspect orientation. The prototype was developed using AspectJ to evaluate the aspect-oriented paradigm. Furthermore, for comparative purposes, the proof-of-concept prototype was developed using Java as an object-oriented language and it focused specifically on the **Usage Decision Facility** of the OAC(UCON) model. Section 8.3 is a formalisation and consolidation of an earlier publication by Padayachee and Eloff(2009).

8.3 Implementation of the proof-of-concept prototype

Figure 8-1 shows the activity diagram for the OAC(UCON) model. In terms of the formal specification presented in Section 7.4.1, this relates to the pre-authorization and ongoing authorisations as well as to the post-authorization formulations.

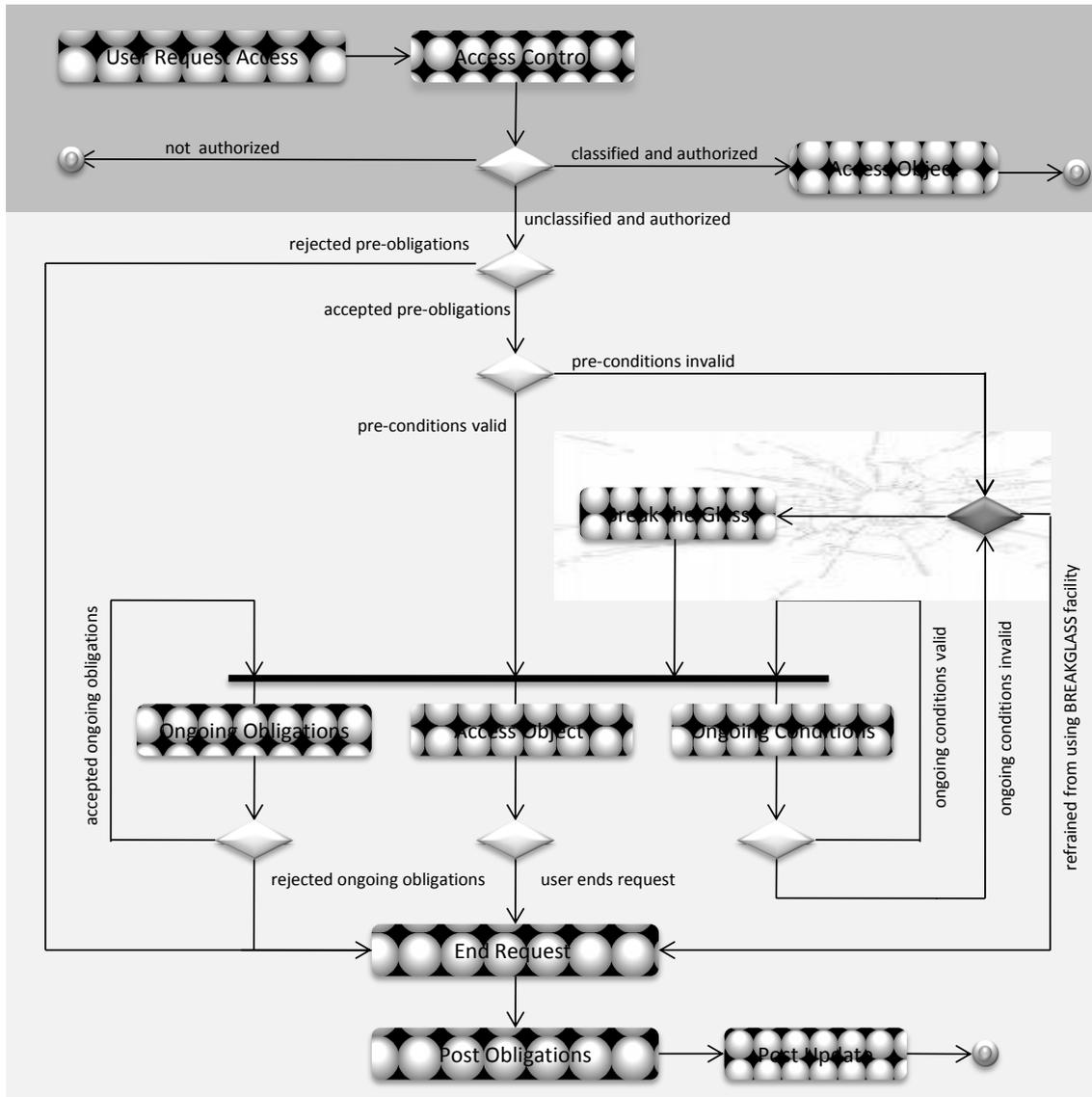


Figure 8-1: State Activity diagram of OAC(UCON) Model)

When a user requests access to an object, authorisation is performed by utilising subject and object information (attributes). Usage rules are used to check whether the request is permissible and whether the data is classified and subject to access control. If the data is in the public domain and hence unprotected by access controls, then the usage deterrence

mechanisms are deployed. Otherwise, the access control proceeds as expected with traditional access control based on user attributes.

Multithreading was used to implement the ongoing authorisations (see Figure 8-2). If a subject requests an object (such as a file), the pre-conditions and pre-obligations are checked and then two separate threads are invoked, representing the ongoing conditions and ongoing obligations respectively.

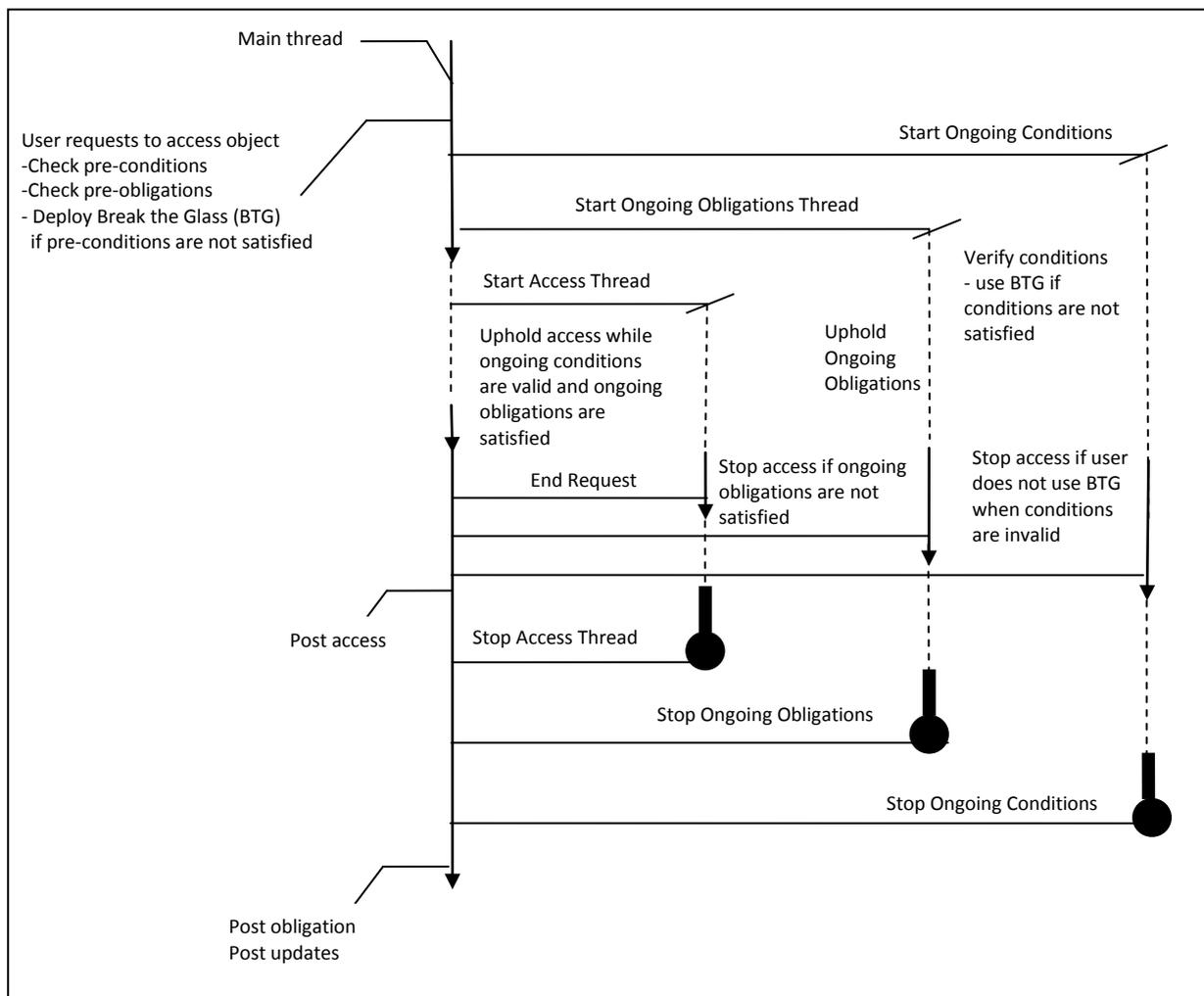


Figure 8-2: Thread Diagram of the OAC(UCON) model

As this model is based on optimistic access control, most users are trusted to access information that is relevant to their context. However, owing to the openness of the architecture, the user has to meet all pre-obligations and ongoing obligations. As the user is expected to behave in a trustworthy manner, the trust is maintained by the user's acceptance of the pre-obligations and ongoing obligations that are coupled with accessing the information. However, if the pre-conditions or ongoing conditions are invalid, the user is allowed to deploy the Break-the-Glass facility to access the information.

If the user does not accept the pre-obligations, he/she is not allowed to access the information at all. If the user does not accept the ongoing obligations, access is revoked. These refusals are not considered as breaches of trust. On the other hand, if the user accessed the information and then refuses to accept the post-obligations, such refusal is considered to be a breach of trust.

In the previous chapter, aspect-orientation was proposed as means of implementing usage control without making invasive changes to the code base of a fully operational system. To illustrate the utility of the aspect-orientation in terms of the enforcement of usage control, a scenario is considered where a fully operational software system that performs traditional access control has to be complemented with the features offered by the OAC(UCON) model during post-delivery maintenance. In this scenario, the class `SampleAuthorization`, as shown in Program Listing 8-1, controls user accesses to objects by using traditional access controls. An object could be a file 'c:\candidate.txt'.

Program Listing 8-1: 'SampleAuthorization' class

```
public class SampleAuthorization implements PrivilegedAction {  
  
    public Object run() {  
        .....  
        AccessObject Access (SubjectName, ObjectName, "READ");  
        AccessObject.request ();  
        .....  
    }  
}
```

As shown in Program Listing 8-1, the execution point where the `request()` method is called involves checking whether this access request by a subject to an object is permissible. The operational system has classes that represent the access and access information called `Access` and `AccessInformation` respectively, where the `Access` class extends the `AccessInformation` class. The `AccessInformation` class represents the nature of the access, in terms of the subject-object attributes and the type of access. The Unified Modelling Language (UML) diagram provided in Figure 8-3 shows how these classes may be integrated with usage control features without modifying the operational system.

Since Aspect-Oriented Programming is a relatively new paradigm, there are no specific standards available in terms of designing aspects with regard to the UML. A position paper by Groher and Schulze (2003) was used to produce the UML diagram (Figure 8-3). The symbol  was created to show an intertype declaration.

As is shown, the Ongoing Obligations, Ongoing Conditions and Break-the-Glass mechanisms are represented by classes `OngoingObligation`, `OngoingCondition`, `BreakTheGlass`; and would inherit from the `AccessInformation` class as these classes require information about the access. The `UsageControlInjector` collaborates with these classes to perform its usage control functionality when a subject requests access to an object. As the `Access` class was not intended to cater for instances where access needs to be revoked owing to invalid usage control obligations and conditions, an `intertypeTypeInjectorOnAccess` aspect was included to perform the requisite actions when access needs to be terminated. For the sake of readability, only core classes that are affected by the aspects are shown. The `Sampleauthorization` class is not shown in the diagram as the `UsageControlInjector` provides its supplemental functionality around the `request()` method found in the `Access` class which is actually responsible for mediating a request between a subject and an object.

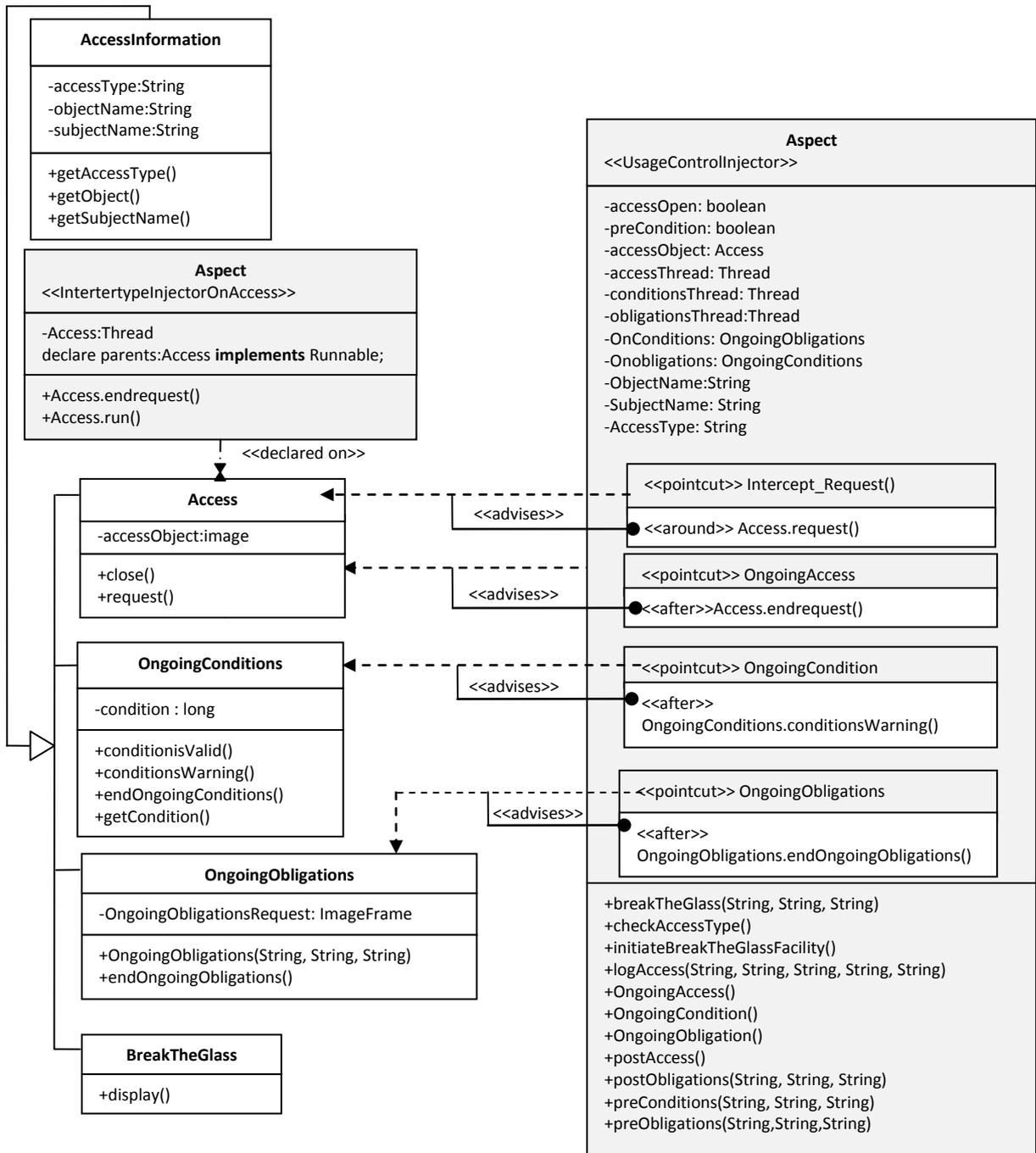


Figure 8-3: UML Diagram showing Aspect UsageControlInjector and Core Classes

8.4 An implementation overview of the proof-of-concept prototype

With aspect-oriented programming, the existing authorisation module which has methods for traditional access control can be augmented with usage control without modifying the source code. Furthermore, all details relating to usage control can be confined in a singular modular structure, namely an aspect, without it being mixed in with this module. To accomplish this, a generic aspect `UsageControlInjector` that delineates four pointcuts was defined. The first pointcut `Intercept_Request` intercepts those calls where a user requests access to an object, i.e. during programs execution where the method body of `request()` executes. The naming of the request method may differ from application to application. The `!within` expression is used to prevent infinite recursive calls. If no special precautions are taken, aspects that advise other aspects can easily and unintentionally advise each other recursively. The `target` keyword has to ensure that the executing code belongs specifically to class `Access`. The *around* advice defines code that is executed around the `request()` method when it is called. This advice initially determines if this information is in the public domain (i.e. controlled by Optimistic Access Control), otherwise it allows traditional access control to proceed as usual. The advice also contains operations to test the `preObligations` and `preConditions`. If the `preObligations` are not satisfied, the user is not allowed to access the object. If the `preConditions` are invalid, the user has the opportunity to use the Break-the-Glass facility to access the feature. The aspect invokes threads to maintain the ongoing conditions and ongoing obligations to control the request to use the object.

Program Listing 8-2: Showing the UsageControlInjector Aspect

```
public aspect UsageControlInjector {
    private static boolean accessOpen;
    private static boolean preCondition = true;
    private Thread obligationsThread;
    private Thread conditionsThread;
    private Thread accessThread;
    private Access accessObject;
    private OngoingConditions OnConditions;
    private OngoingObligations Onobligations;
    private String SubjectName;
    private String ObjectName;
    private String AccessType;

    pointcut Intercept_Request(Access AccessObject) :
execution(* *.request(..) && !within(UsageControlInjector)
&& target(AccessObject) ;
    void around (Access AccessObject )
    : Intercept_Request(AccessObject){
        accessOpen = true;
        accessObject = AccessObject;
        SubjectName = accessObject.getSubjectName();
        ObjectName = accessObject.getObject();
        AccessType = accessObject.getAccessType();
        if (OptimisticRights()){
            if (preObligations(SubjectName, ObjectName, AccessType)){
                if (preConditions(SubjectName, ObjectName, AccessType)
                    || breakTheGlass(SubjectName, ObjectName, AccessType)){
                    //start access Thread
                    //start OngoingObligation Thread
                    //start OngoingCondition Thread
                    while(accessOpen){
                        //wait
                    }

                    postObligations(SubjectName, ObjectName, AccessType);
                }
            }
        }
    }

    public boolean checkAccessType(){
        //determine whether this information is subject to optimistic access
        control
        return true;
    }

    //when there is conditions warning
pointcut OngoingCondition(): call(* *.conditionsWarning(..) ) &&
target(OngoingConditions);
after(): OngoingCondition(){
    initiateBreakTheGlassFacility();
}

    // when the user ends ongoingObligations
pointcut OngoingObligation() : call(* *.endOngoingObligations(..) ) &&
target(OngoingObligations) && !within(UsageControlInjector);
after(): OngoingObligation(){
    postAccess();
}
}
```

```
// when the access ends
pointcut OngoingAccess() : call(* *.endrequest(..) ) && target(Access) &&
!within(UsageControlInjector);
after(): OngoingAccess(){
    postAccess();
}

void initiateBreakTheGlassFacility(){
//method to initiate the Break-the-Glass facility
}

boolean preObligations(String SubjectName, String ObjectName, String
AccessType){
//method to perform preObligations
}

boolean preConditions(String SubjectName, String ObjectName, String
AccessType){
//method to perform preconditions
}

void postAccess(){
//clear up
//update logs
}

public void postObligations(String SubjectName, String ObjectName, String
AccessType){
//method to perform postObligations
}

boolean BreakTheGlass(String SubjectName, String ObjectName, String
AccessType){
//perform Break-the-Glass facility
}

void logAccess(String SubjectName, String ObjectName, String AccesType,
String Notice, String RedFlag){
//write to log file
}
}
```

The next two pointcuts, `pointcut OngoingCondition` and `pointcut OngoingObligation`, intercept execution points that indicate that the ongoing conditions and ongoing obligations are no longer satisfied. The *after* advices of each pointcut define code that is executed after such an irregularity is detected. In this case, if some action results in the `ConditionsWarning` or an `endObligations` method being called on either the `conditions` object or the `obligations` object, then this call will be intercepted by these pointcuts. If the ongoing obligations are no longer satisfied, then the access is revoked immediately. If the ongoing conditions are no longer satisfied, the user has the opportunity to use the Break-the-Glass facility to continue with the access.

The last pointcut is used to intercept execution points where the access is terminated by the user. Subsequently post-access activities such as logging the access are performed and the Ongoing Conditions and Obligations threads are ceased. (Appendix F provides details on the AspectJ semantics.)

The `Access` class would require additional functionality to mediate users' requests to an object and to end requests when required. This functionality may not have been provided in a way that fits in with the current usage of the `Access` class. To maintain the integrity of the class, aspect orientation permits a seamless integration of this additional functionality by facilitating the creation of a special aspect known as an *intertype declaration* without modifying the `Access` class. The *intertype declaration* construct is supported by aspect-oriented programming languages such as AspectJ. An *intertype declaration* is generally used to add on information such as methods or fields to an object without modifying the existing class. Furthermore, as this process needs to be controlled within thread, in Java it implies that this class must implement the `java.lang.Runnable` thread interface. With AspectJ, this can be done using the `declare parents` syntax so that `Access` class can be an active object. The relevant classes are provided in Appendix C.

Program Listing 8-3: Depicting an InterTypeDeclaration Aspect

```
public aspect IntertypeDeclarationOnAccess {
    declare parents: Access implements Runnable;
    Thread Access.aThread;
    public void Access.endrequest() {
        aThread = null;
        close();
    }
    public void Access.run() {
        //perform the request
    }
}
```

The `UsagecontrolInjector` is relatively generic as it can be reused within other contexts as well. Only the method specified that performs the access control, namely `request()`, would have to be re-specified according to the system naming. The `IntertypeDeclarationonAccess` aspect is partially generic, as the class name

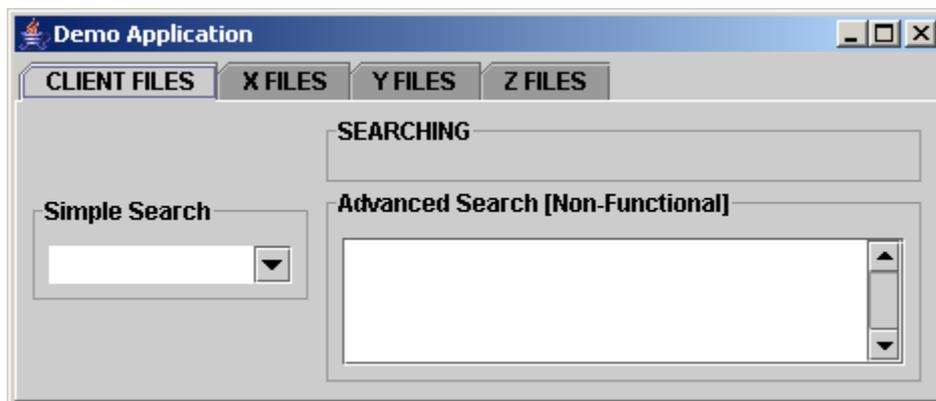
Access would have to be re-specified accordingly. Unlike typical instrumentation, using the aspect-oriented paradigm results in better consistency, as all methods that match the pointcuts are identified. Evidently, it is possible to augment usage control features in a system without modifying the existing system.

8.5 Proof-of-concept prototype operation

Step 1: Login and authentication of the user



Step 2: An application interface appears which allows searching of files.



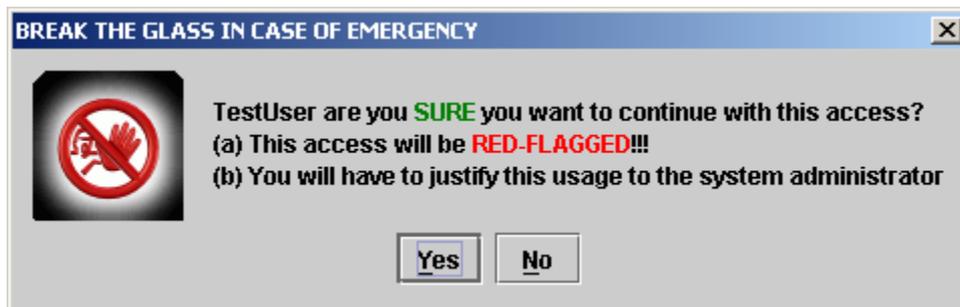
Step 3: User selects a file to open and the pre-obligation dialogue box opens. User has to accept the pre-obligation to move on to the next step.



Step 4: Pre-conditions warning appears if the pre-conditions are NOT met.



Step 5: The Break-the-Glass facility is invoked. User has to accept the Break-the-Glass option in order to access the file.



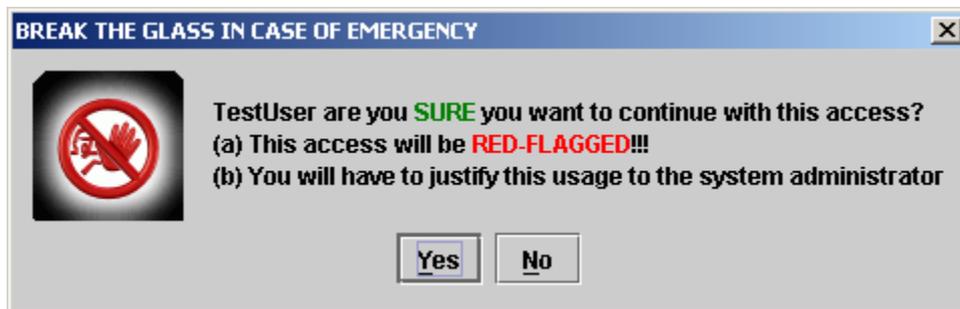
Step 6: Ongoing Obligations pop up while user accesses the file. The ongoing obligations window on the right is an example of an ongoing obligation that involves presenting the user with the security policies related to the file being accessed.



Step 7: Ongoing Conditions pop up intermittently if the ongoing conditions are not met whilst the user is accessing the file.

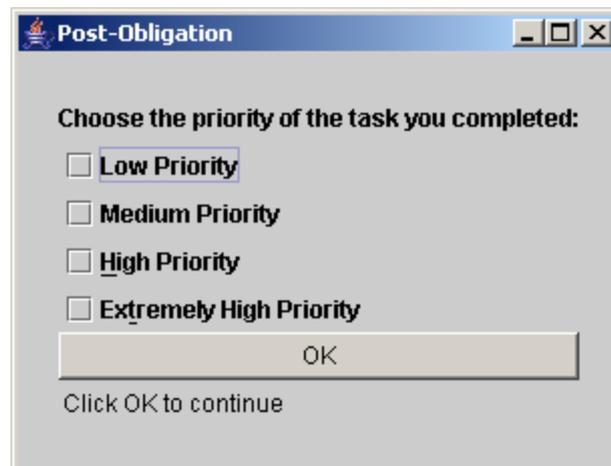


Step 8: The Break-the-Glass facility is invoked.

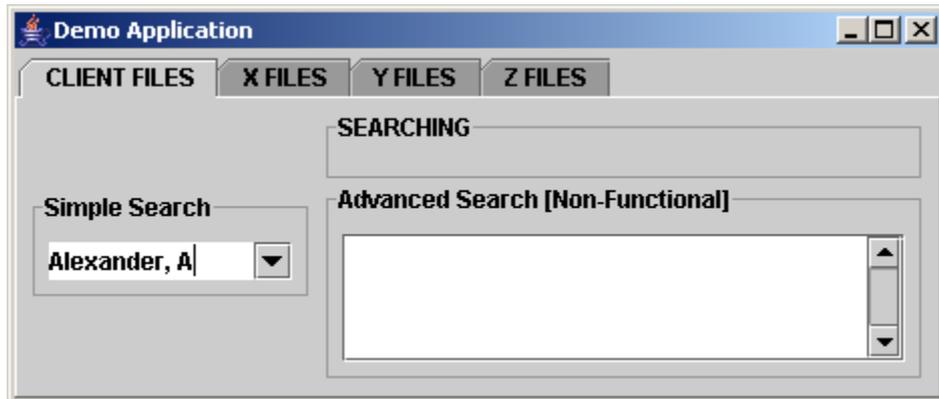


Step 9: User is allowed to sustain the access by accepting the Break-the-Glass option.

Step 10: Post-Obligations are invoked once the user closes the file concerned. This is an example of a post-obligation that is used to assess the user's trustworthiness.



Step 11: Back to Original Interface



8.6 Evaluation of the Aspect-Oriented Approach

In this section, the aspect-oriented approach is evaluated against the object-oriented approach. An object-oriented version of the proof-of-concept prototype was developed for comparative purposes.

8.6.1 The Design Approach

The UML diagram of the Object-Oriented Version is presented in Figure 8-4 below. Compared to the Aspect-Oriented Approach presented in Figure 8-3, it can be observed that the coupling between classes in the system have increased. It is not always clear how to best measure a metric such as coupling in an aspect-oriented system and how to compare it to its equivalent in a corresponding object-oriented system; as yet there does not exist a definitive work for metrics for aspect-oriented systems. However, it is still possible to observe fine-grained changes in coupling by reasoning about the changes in the code base (Singh, 2005).

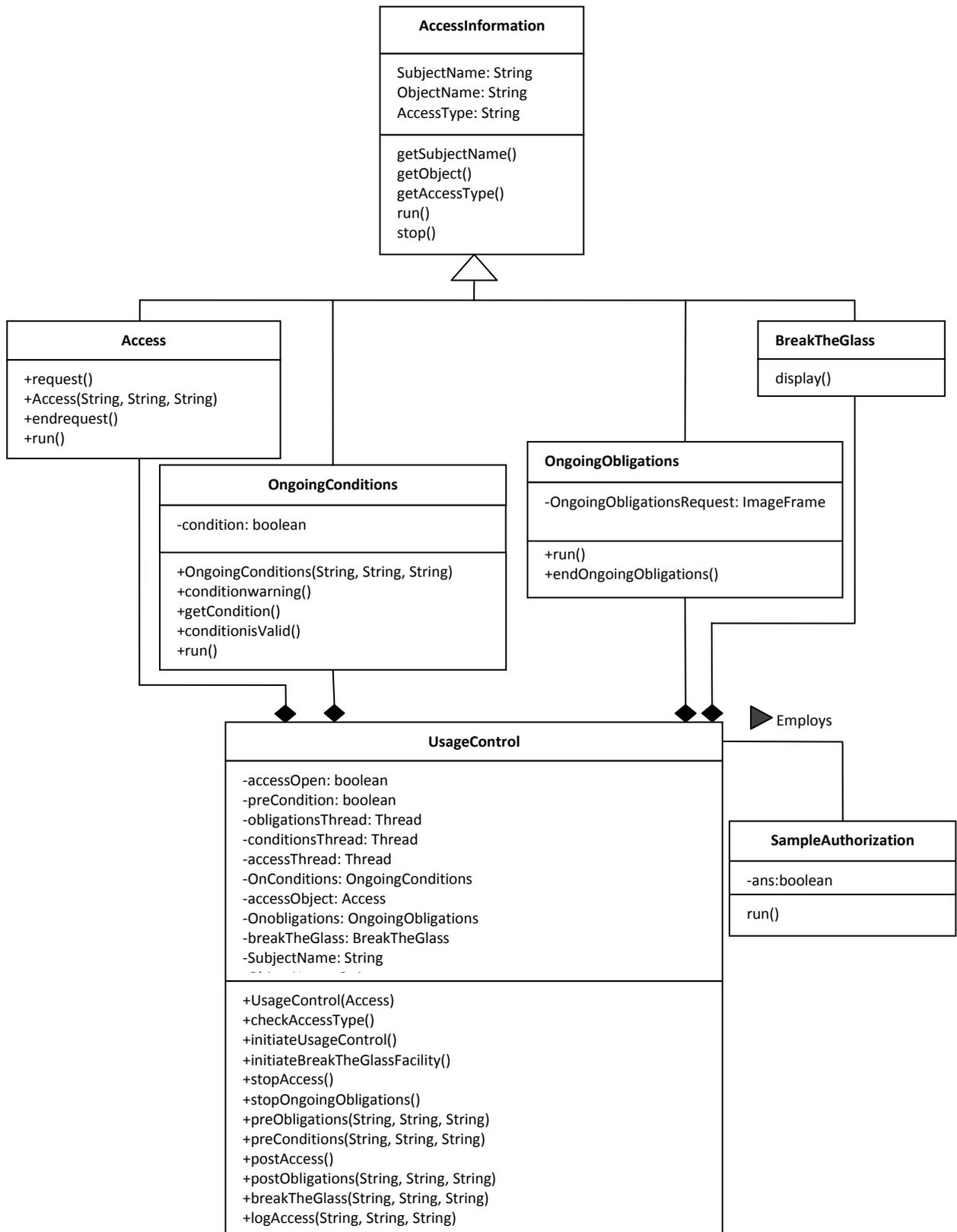


Figure 8-4: Showing the OOP UML of Core Classes

Compared to the aspect-oriented version, there appears to be a reduction in the scattering of concerns (see Figure 8-5) at the package level. For example, the aspect-oriented version does not cross-cut the class responsible for authorisation (i.e. `AuthorizationSim` in the diagram). In the case of the object-oriented version, the `Usagecontrol` class has to interact with the `Access`, `OngoingObligations` and `OngoingConditions` classes, as the class has to be aware of the fact that the conditions are no longer met; that the `OngoingObligations` are not being fulfilled; or that the user has terminated the request. In the case of the aspect-oriented version, the `UsageControlInjector` aspect observes each of these objects and decides what action to perform. It has been calculated that the usage control function is scattered across four classes in the object-oriented version.

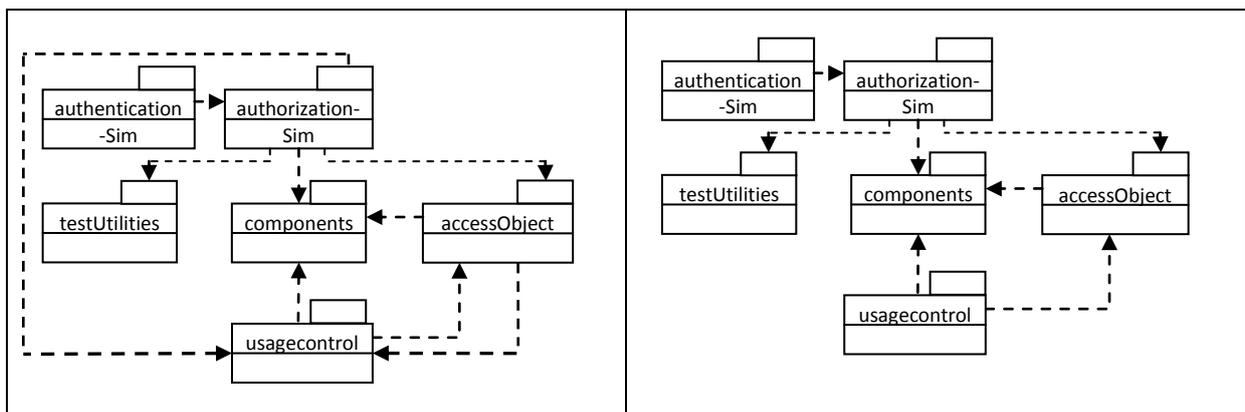


Figure 8-5: OOP package level diagram vs AOP package level diagram (on the right)

8.6.2 Execution Time and Memory Usage

Figure 8-6 shows the change in execution time. In the bar graph, the upper bar represents the time used by the aspect-oriented system while the lower bar represents the quantity for the object-oriented system. The values for these evaluations are calculated by averaging the data of several test runs. It can be noted that the object-oriented version is 1.9% faster than the aspect-oriented version, a difference that is actually negligible. Several reasons could account for this, such as user speed and the speed of the computer processor used for the experiments.

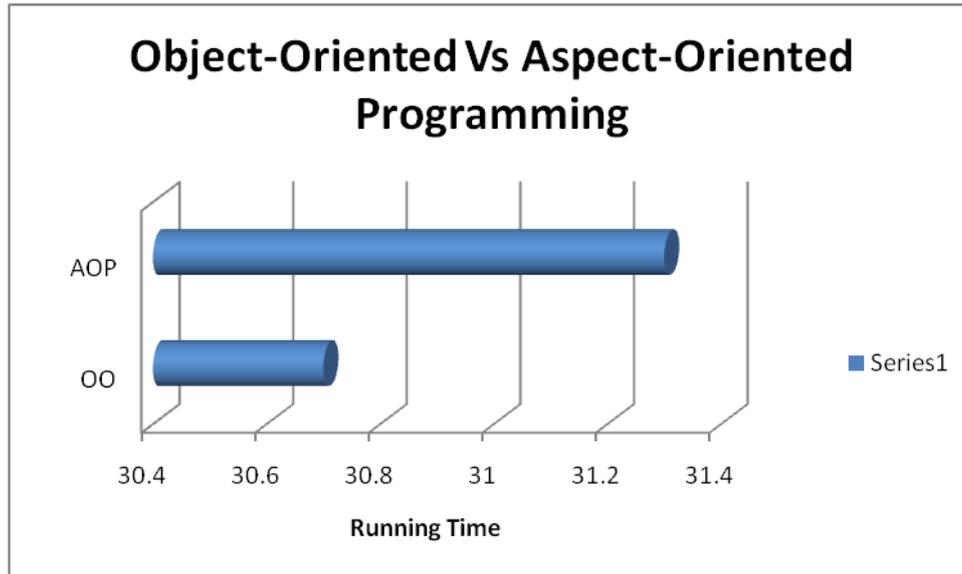


Figure 8-6: Showing comparisons of the execution time of OO vs AOP

Next, the amount of memory (Figure 8-7) that the Java Virtual Machine had demanded from the operating system at the end of each test run was compared. The object-oriented version used only 1.9% less memory than the aspect-oriented version. This figure is within a reasonable margin of error. The results of the tests, which were conducted on an Intel(R) Core 2 Duo CPU E6850 with 3.00 GHz and 1.96 GB of RAM, show that aspect-oriented programming can compete with object-oriented version.

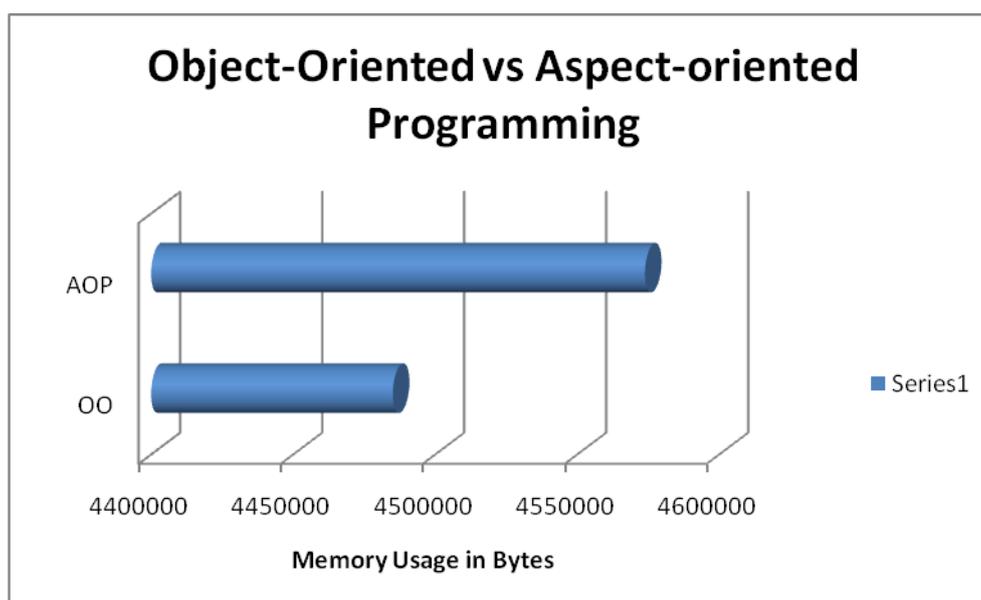


Figure 8-7: Showing comparisons of and Memory Usage of OO vs AOP

8.7 Evaluation of the model concept

The design science research methodology was used to conduct a small-scale experiment based on the following activities: build, evaluate, theorise, and justify (March and Smith, 1995). The experiment involved a problem identification stage (which was done in Chapter 1), design and development of prototypes stages, and an evaluation (Offerman et al., 2009) stage. The participants who were involved in the design and implementation of evaluative prototypes were Computer Science Honours students from the University of Pretoria. The concept specification was scaled up to a real-world scenario and included a mixed-initiative access control framework together with trust (see Appendix D). The purpose of this process was to identify if there were any vacuities, ambiguities or inconsistencies in the model concept. During the evaluation stage, the participants interacted with the evaluative prototypes and provided value judgements on it in terms of the efficacy of the security mechanism provided by the product concept.

A small segment of the evaluation involved the usability of the security mechanisms provided by the model concept. According to Jøsang and Patton (2001), security usability is concerned with the study of how security information should be handled in the user interface. In this context, the usability of the security mechanisms was evaluated. According to Whitten and Tygar (1999) security software is usable if the people who are expected to use it

- are reliably and made aware of the security tasks they need to perform;
- are able to figure out how to successfully perform those tasks;
- do not make dangerous errors; and
- are sufficiently comfortable with the interface to continue using it.

Qualitative data collections were employed, namely participant observation and open-ended interviewing (see Appendix D):

Observation: The idea with participant observation was to determine whether the end-user can successfully complete a task relating to the evaluative prototypes.

Qualitative interview: A qualitative, open-ended interview was conducted to determine the participants' perceptions of the appeal of the model concept in terms of data misuse.

Participants had to address the following in terms of the model concept: Weaknesses, Strengths, Potential improvements, Viability, Applicability and Scalability.

To facilitate the process, the issues concerning the evaluation were formulated into 11 statements. The participants then provided a judgment on each statement. The following data was gathered from the experiment and are discussed in the paragraphs that follow:

Statement 1: The product specifications as given in the assignment were ambiguous and incomplete.

Statement 2: The product specifications as given in the assignment could easily be translated into an implementable product.

Both statements above focused on the viability (or not) of the product, and it was found that 78% of the participants judged the specifications to be unambiguous and complete. Two participants stated that the notion of priorities of tasks needed to be addressed, as the priorities of tasks were assigned randomly in the specification. The priority of the task and the use of the Break-the-Glass feature were examined to determine whether the user utilised the Break-the-Glass facility for a bona fide emergency. If this priority of task did not warrant 'breaking the glass', then the user's rights to information under the optimistic access control domain were constrained. Three of the participants in the study felt that the specifications given were ambiguous and incomplete, and claimed that this had led to misinterpretation. Another participant claimed that the specification "did not give explicit rules for the break-the-glass". All participants nevertheless agreed that the specifications could easily be translated into an implementable product. In fact, one participant indicated that the specifications were easy to divide into implementable components. The product concept was judged to be highly viable and the participants were able to implement it using several approaches, including Java, C# and PHP.

Statement 3: In terms of the enforcement of security, other mechanisms such as a written policy document or adequate training would have been more effective than the mechanisms identified in the product concept.

With regard to the effectiveness of the product concept in relation to other non-technical approaches, 78,5% of the participants disagreed that other mechanisms such as training would have been more effective than the product concept. Three participants felt that other mechanisms – in combination – would increase the security overall, while four others felt that the training and policy documents were "simpler to ignore" and "not a constant reminder" as was the case with an automated system. In addition, the prototype concept enabled the tracking of a user's actions.

Statement 4: The flexibility offered under the optimistic access control domain is a security risk.

Based on the risk of using optimistic access control (owing to its flexibility), 78,5% agreed that optimistic access control was a security risk and that data should be protected by other means. However, some participants indicated that it depended on the nature of the organisation and its data, and that some environments such as the medical industry actually required the proposed level of flexibility.

Statement 5: Specifying system conditions, such as limiting access according to the time-of-day, may deter users from abusing their privileges.

Altogether 78.5% of the participants agreed that specifying conditions would deter users from abusing their privileges. Most participants felt that these conditions would give the user the feeling that they were "doing something wrong" and that they would be deterred as a result. They also felt that the threat of punishment and losing trust might provide a motivation for users not to abuse their privileges.

Statement 6: The 'break-the-glass' facility is vulnerable to abuse.

In terms of the susceptibility of the break the glass facility 71% of the participants agreed that the Break-the-Glass system was vulnerable to abuse. However, most of them indicated that the threat of being discovered after the event was a way of preventing the Break-the-Glass facility from being misused.

Statement 7: The protection mechanisms, such as fulfilling obligations, will compel users to comply with the established rules of behaviour in order to protect confidential information.

In terms of satisfying obligations, 85% of the participants agreed that the fulfilment of obligations would compel users to comply with the established rules of behaviour. Using obligations would prevent users from claiming ignorance as an excuse for not complying. Furthermore, since users are intimidated by warnings, user responsibility could be expected to increase.

Statement 8: An individual who interacts with the system will recognize that access is dependent on user responsibility as well as technical access control.

With regard to security usability, 71.4 % of the participants agreed that an individual who interacted with the system would recognise that access was dependent on user responsibility as well as technical access control. Those participants who opposed the statement argued that users were irresponsible and untrustworthy.

Statement 9: The risk of losing one's rights to information under the optimistic access control domain may deter one from abusing one's privileges.

Due to the severity of punishment, 84.6% agreed that the risk of losing one's rights to information under the optimistic access control domain might act as a deterrent against abusing one's privileges. The threat of being caught and losing one's trust was a strong motivator. However, participants agreed that if the user's premeditated goal was to steal data, these mechanisms would not prevent such incidences.

Statement 10: The conditions, obligations and the break-the-glass mechanisms may be distracting to a user.

In terms of security usability, 57% of the participants disagreed that the conditions, obligations and the Break-the-Glass mechanism would be distracting to the user. Even those users who agreed felt that after some time most users would ignore these pop-ups anyway. However, all of this would depend on how the user interface was designed. It was important to be presented in such a way that users should not become complacent or exasperated about the messages.

Statement 11: Most users will ignore the messages about conditions and obligations relating to the access.

Half of the participants agreed that users would ignore the messages about conditions and obligations relating to the access. They felt that, in time, users would eventually pay no attention to these messages. The other 50% of participants, who disagreed, proposed that users should be forced to respond to the message. Furthermore, participants posited that users would ignore these messages unless the consequences were clearly specified.

Although most participants regarded optimistic access control as a security risk, participants reasoned that the additional facilities of obligations and conditions might deter users from abusing their privileges. Participants suggested that constant reminders would ensure that users would not perform illegitimate actions seeing that they would be monitored. Some participants indicated that the separation of public domain information from the private domain was strength, as it allowed for information to be subject to different controls. Furthermore, it was quite a simple task to maintain the access control policies for information under the optimistic access control domain. Regarding improvements, it was suggested that the conditions should be more dynamic and that they should be based on user profiles. Participants also suggested that rather than displaying pop-ups for every access, a single-sign or a pop-up should be flashed intermittently. Regarding weaknesses, the participants felt that users might try to bypass warning messages because they were annoyed by them and that there was too much reliance on the trustworthiness of users.

The evaluation exercise revealed that the model concept could be appropriate to call centres, dynamic environments, medical information systems and Wikipedia. It was reasoned that the system would be relevant in situations where users were transitory. This kind of system would also be more fitting for users who were professionals rather than the average user. It would furthermore be more suitable in environments where damage was reversible or in small organisations that used data that was not that sensitive.

Augmenting traditional access control with usage control features is expected to slow down program execution, as it involves the inclusion of additional code in the functional system. In terms of security usability, controls such as pre-obligations and ongoing obligations may be distracting and impact negatively on the productivity of users. Perhaps, as the user becomes more 'trustworthy', some obligations or conditions may be relaxed or negotiated. The costs of implementing usage control as a deterrent may have to be weighed up against the cost of information misuse. South Africa's draft bill on the protection of personal information is viewed as a means to ensure South Africa's future participation in the information market by providing 'adequate' information protection of an international standard (see (*CHAPTER 9: A DRAFT BILL ON THE PROTECTION OF PERSONAL INFORMATION, 2005*)). If individuals are ensured that their privacy is taken into account in a software system, it is understandable that they will trust the system with their private information. The survival of e-business will probably depend on its ability to guarantee the privacy of its clients.

The proposed OAC(UCON) model does not account for trust issues; thus this needs to be addressed in future renditions of the model. In addition, the relationship between using the Break-the-Glass facility and the priority of the task needs to be explored. Since using this facility is dependent on the urgency of the task, the rules governing the Break-the-Glass facility need to be defined in more detail.

8.8 Conclusion

The aspect designed for the enhancement of optimistic access control was tested in terms of a proof-of-concept prototype. It was found that confining all the operations relating to usage control to a single modular structure would reduce both development and maintenance costs. Next, the relationship between multithreading and cross-cutting behaviour was explored in this chapter. It was shown that the aspect-oriented approach does not impact significantly on execution time or memory usage and that aspect-oriented programming introduced fewer scattering of usage control concerns and less coupling between classes.

Owing to the sample size which was quite modest (i.e. 14 participants), the limitations of the experiment need to be taken into account when making generalisations from the research. The nature of the study required participants to be competent at programming a large system independently and to deliver the product within a reasonable time frame. Purposive sampling had to be employed as the participant had to be an advanced programmer who also had the time available to do the task. These two requirements were met by the students enrolled for the Computer Science Honours programme at the University of Pretoria. It is difficult to find members of society who would fit this unique profile. The other limitation posed by the research method was that every participant's final product had to be evaluated and the participants had to share their insights on the model as well as their design decisions. Having a large sample would make this task extremely time consuming. It would also imply that each participant's involvement would have been superficial. A small sample, on the other hand, allowed for a more in-depth analysis of each participant's value judgement. A future study may involve replicating the research method with a new group of students from another university.

The model has not been tested within a large distributed system with several end-users in an organisational setting. However, participants who tested the model concept can be considered the representatives of stakeholders in the information technology industry. As postgraduate students, they have extensive knowledge of information systems and are



currently employable or employed within the information systems sector. The product was found to be highly viable as all participants were able to implement the scaled-up version of the concept. The usability of the system was reasonable, except for the criticism that the usage control features might be distracting and could eventually be ignored. However, the evaluation revealed that users would understand that access control was based on technical control as well as user responsibility. The effectiveness of optimistic access control was found to be largely dependent on the usage control features, as optimistic access control on its own posed a security risk. The evaluation nevertheless proved that by employing usage control features of obligations and conditions, this risk would be reduced.

CHAPTER 9:

CONCLUSION

9.1 Introduction

The study in hand focused on a model for usage control under the optimistic access control paradigm, i.e. the OAC(UCON) model. To increase the applicability of the model, it was presented within a mixed-initiative access control framework. The pragmatic issue of implementing such a model within the wider context of access control formed the topic of discussion in this thesis. To ease the integration of the proposed model into an existing access control framework, an aspect-oriented approach was selected. The motivation for this study was posited in Chapter 1 and required a number of research goals to be addressed. In this closing chapter the researcher evaluates the extent to which the objectives of the research goals have been met. Finally, it concludes with a discussion of the main contribution of the research and suggestions for further research.

9.2 Main contribution

This research did not promote the notion that traditional access control models were inferior to optimistic access control. Rather, it suggested that the two approaches might work well in a mixed-initiative approach. The OAC(UCON) model is flexible and reduces the burden of setting pre-configured security policies for every subject-object relationship, and thereby reduces the load on system administrators. However, the model acknowledges that the gains realised by flexibility should not be negated through data misuse. Thus, the model provided sufficient deterrents against data misuse by leveraging the security mechanisms

offered by usage control. It was suggested that data that cannot be reasonably protected within traditional access control could be protected by these usage control deterrents.

As was stated earlier, the proposed solution could well ease the burden of system administrators significantly. It is rather difficult for administrators to predict all of the possible usage scenarios and thus all of the necessary permissions. With optimistic access control, it is ultimately left to the users to make that judgement. Consequently, the complexity of implementing and maintaining pre-configured access control policies is shifted to the way the user interacts with the system. Adapting usage control as a deterrent provides a proactive mechanism over and above the retroactive methods of auditing and accountability. By using the OAC(UCON) model, a larger subset of information may be relegated into the public domain.

This research also addresses the issue of continuity within usage control and its practical implementation within the access control context rather than within the digital rights management context. The thesis is consequently presenting pragmatic ways of introducing continuity within the access control dimension. In terms of the proof-of-concept that was developed, the ongoing obligations involved presenting the user with the relevant security policies while he/she accesses the related information. This is an example of the type of application that educates the user on approved security policies with regard to the specific data that he/she is interacting with.

Investigating the efficacy of the aspect-oriented programming language can be considered one of the major contributions of this research. It was found that usage control can be completely separated from access control and other application logic. It was also determined that the performance differences between the object-oriented and aspect-oriented version were negligible. Additionally, there was less coupling between classes with the aspect-oriented version, which increased the readability and understandability of the code. The relationship between multithreading and cross-cutting behaviour was also explored and the study demonstrated how ongoing authorisations could be maintained with multithreading.

The model is unique in that the access controls are applied in the application layer. It provides supplemental usage control to objects that have their access rights defined within the database layer. The rights defined in the database layer may be relaxed in the application layer or maintained as specified. If the rights were relegated to optimistic rights, then the rights are relaxed and supplemented with optimistic rights. Alternatively, if these rights were considered to be highly classified, then the OAC(UCON) allowed these rights to remain as specified.

9.3 Revisiting the problem statement

The problem statement highlighted the inadequacies of current access control models, namely their lack of flexibility and difficulties in assigning pre-configured access control policies. To this end, a critical overview of popular access control models was provided and an optimistic access control model was recommended as a means of correcting these deficiencies. Since it was noted that optimistic access control is far too flexible to be used in practice, it was enhanced with usage control in order to offer greater rigour. In this model, usage control was reformulated under optimistic access control to act as a mechanism for deterrence rather for denial of access. Thus the OAC(UCON) model was developed. To improve its general applicability, it was presented in a mixed-initiative access control framework, where pessimistic access control models were complemented with optimistic access control models. In order for this type of integration to be successful, a software approach was inferred that would allow for the seamless augmentation of traditional access control with optimistic access control enhanced with usage control, namely the aspect-oriented approach. A partially generic usage control aspect was presented that could, in theory and with minor modifications, be augmented seamlessly into a fully operational system. The aspect-oriented approach was also evaluated in terms of performance against an object-oriented approach. Finally, the design science research methodology was employed to test the model concept and to assess its scalability with other access control measures and within a wider context so as to gain insight into the usability of the model concept.

9.4 Future Research Directions

The element of trust within the OAC(UCON) model warrants an investigation into human behaviours and the responses to its application. It would be pragmatic to investigate whether the model concept in fact dissuades individuals from accessing and misusing information in the public domain. Future research could be directed at the inclusion of trust-based mechanisms to update a user's optimistic rights. Presently the model does not account for how trust levels may change when a user loses his/her optimistic rights. In order to test the scalability of the model concept, the notion of trust needs to be considered and its inclusion would complete the mixed-initiative access control framework. The issue of trust was considered in terms of how a user's rights to information under the optimistic paradigm may be modified based on prior usage. In the case of the evaluative prototype of the model it was presumed that, at the onset, each user had access to optimistic rights rated as 'high'. However, as the user demonstrated his/her untrustworthiness, the level of access was downgraded to 'medium' and finally to 'low'. As their optimistic rights were demoted, the view to information became increasingly constrained. The users' optimistic rights were updated using fuzzy logic. Future research could well involve considering the factors that influence trust levels. In the specification given to the participants as part of the design research method, the priority of the task and the user's previous trust level were used to update his/her optimistic rights in a fuzzy matrix.

An alternative research direction may involve investigating whether the model concept increases the propensity towards compliant information security behaviour. This refers to a set of core information security activities that has to be carried out by end-users so as to maintain information security as defined by information security policies (Chan et al., 2006).

It is also suggested that future studies should involve a case study to test the usability of the aspect-oriented approach since it has not been tested in an organisational context yet. However, confining all the operations pertaining to usage control to a single modular structure will alleviate both development and maintenance costs as it can be integrated seamlessly into a system based on traditional access control.

9.5 Conclusion

The proposed solution to access control draws inspiration from some of the principles advocated by agile methods. For example, consider the agile principles relating to embracing change and maintaining simplicity. In the case in hand access control was implemented in its most rudimentary form. As with agile methods, the reliance was on people rather than on complicated processes to maintain control.

The viability of the model concept was demonstrated in a scaled-up version where it was possible to create a mixed-initiative access control model. It was found that optimistic access control is a security risk, but that the combination of usage control features coupled with monitoring and punitive action may deter users from abusing their privileges. The security usability aspect of the concept would need to be improved, as users would probably sooner or later disregard the obligations and conditions. Accordingly these notions needed to be more dynamic and responsive. The obligations and conditions messages need to be updated constantly and they have to be reformulated to retain a user's focus. The evaluation revealed that the concept may be appropriate to call centres, medical information systems, temporal environments and smaller organisations where data is not viewed as particularly sensitive. This kind of system would also be more appropriate for users who have a degree of professionalism more so than the average user and in environments where damage is reversible.

The use of aspect-oriented programming contributed to the principles of embracing change and maintaining simplicity. Adapting usage control as a deterrent provides a proactive mechanism over and above the retroactive methods of auditing, accountability and recoverability. It is envisioned that a larger subset of information may be transferred to the public domain, thus obviating the need for specifying convoluted access control policy decisions.