CONTEXT-AWARE ACCESS CONTROL IN UBIQUITOUS COMPUTING (CRAAC)

A thesis submitted to the University of Manchester
for the degree of Doctor of Philosophy
in the Faculty of Engineering and Physical Sciences

July 2010

By

Ali Ahmed

School of Computer Science

Contents

List of Tables

List of Figures

Abstract

Ubiquitous computing (UbiComp) envisions a new computing environment, where computing devices and related technology are widespread (i.e. everywhere) and services are provided at anytime. The technology is embedded discreetly in the environment to raise users' awareness. UbiComp environments support the proliferation of heterogeneous devices such as embedded computing devices, personal digital assistants (PDAs), wearable computers, mobile phones, laptops, office desktops (PCs), and hardware sensors. These devices may be interconnected by common networks (e.g. wired, wireless), and may have different levels of capabilities (i.e. computational power, storage, power consumption, etc). They are seamlessly integrated and interoperated to provide smart services (i.e. adaptive services). A UbiComp environment provides smart services to users based on the users' and/or system's current contexts. It provides the services to users unobtrusively and in turn the user's interactions with the environment should be as non-intrusive and as transparent as possible. Access to such smart services and devices must be controlled by an effective access control system that adapts its decisions based on the changes in the surrounding contextual information.

This thesis aims at designing an adaptive fine-grained access control solution that seamlessly fits into UbiComp environments. The solution should be flexible in supporting the use of different contextual information and efficient, in terms of access delays, in controlling access to resources with divergent levels of sensitivity.

The main contribution of this thesis is the proposal of the Context-Risk-Aware Access Control (CRAAC) model. CRAAC achieves fine-grained access control based upon the risk level in the underlying access environment and/or the sensitivity level of the requested resource object. CRAAC makes new contributions to the access control field, those include 1) introducing the concept of level of assurance based access control, 2) providing a method to convert the contextual attributes values into the corresponding level of assurance, 3) Proposing two methods to aggregate the set of level of assurance into one requester level of assurance, 4) supporting four modes of working each suits a different application context and/or access control requirements, 5) a comprehensive access control architecture that supports the CRAAC four modes of working, and 6) an evaluation of the CRAAC performance at runtime.

Declaration

No portion of the work referred to in this thesis has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.

Copyright

i. The author of this thesis (including any appendices and/or schedules to this thesis) owns any copyright in it (the "Copyright") and s/he has given The University of Manchester the right to use such Copyright for any administrative, promotional, educational and/or teaching purposes.

ii. Copies of this thesis, either in full or in extracts, may be made only in accordance with the regulations of the John Rylands University Library of Manchester. Details of these regulations may be obtained from the Librarian. This page must form part of any such copies made.

iii. The ownership of any patents, designs, trade marks and any and all other intellectual property rights except for the Copyright (the "Intellectual Property Rights") and any reproductions of copyright works, for example graphs and tables ("Reproductions"), which may be described in this thesis, may not be owned by the author and may be owned by third parties. Such Intellectual Property Rights and Reproductions cannot and must not be made available for use without the prior written permission of the owner(s) of the relevant Intellectual Property Rights and/or Reproductions.

iv. Further information on the conditions under which disclosure, publication and exploitation of this thesis, the Copyright and any Intellectual Property Rights and/or Reproductions described in it may take place is available

from the Head of School of Computer Science (or the Vice-President).

Acknowledgements

First of all, I thank Allah (the lord) for all his blessings, I would not have completed my PhD without his guidance and success. I would like to thank Dr. Ning Zhang for her dedication and support throughout my PhD. Her comments and feedback were invaluable and I really appreciate that. I would like to thank the Cairo University, Egypt, for its financial support. I also would like to thank my friends Tarek, Al-Kateb, and Essam for their help and support. May be last but definitely not least, I really want to thank my wife, Noha, for all the support and kindness that I have been overwhelmed by. Taking care of our kids, Omar and Jojo, has really made me concentrating on my studies and progressing faster. In addition, her understanding of my work and the good technical advices were invaluable.

Dedication

In the name of Allah

Thy Lord hath decreed, that ye worship none save Him, and (that ye show) kindness to parents. If one of them or both of them attain old age with thee, say not "Fie" unto them nor repulse them, but speak unto them a gracious word. (23) And lower unto them the wing of submission through mercy, and say: My Lord! Have mercy on them both as they did care for me when I was little.(24) [Al-Isra Chapter, The Holly Quran]

To those who have been always beside me, supporting me, and encouraging me for no personal benefits

To you mother and father

I love you, I really do ...

Ali

Chapter 1

Introduction

1.1 Project Context

1.1.1 Ubiquitous Computing (UbiComp)

Ubiquitous computing is a new computing paradigm, where computing devices and related technology are widespread (i.e. everywhere) and services are provided at any-time. The technology is embedded discreetly in the environment to raise users' awareness. UbiComp environments support the proliferation of heterogeneous devices such as embedded computing devices, personal digital assistants (PDAs), wearable computers, mobile phones, laptops, office desktops (PCs), and hardware sensors. These devices may be interconnected by common networks (e.g. wired, wireless, etc) [10] and may have different levels of capabilities (i.e. computational power, storage, power consumption, etc). They are seamlessly integrated and interoperated to provide smart services (i.e. adaptive services). A UbiComp environment provides smart services to users [11] based on the users' and/or system's current contexts. For example, in a smart home environment, tasks such as controlling heating and lighting, going to a grocery store, scheduling

home appliances, and cooking are automatically adjusted according to the immediate context. Thus, a resident does not have to physically intervene or think about these tasks [12]. For instance, a smart fridge may alert a resident if it is running out of milk. It may even place an order for milk at a distant grocery, because the fridge 'knows' that it is cheaper than the one nearby. Or, to cite a more serious example, a body sensor that records a low blood pressure reading for a cardiac patient should immediately call an ambulance. A UbiComp environment provides these kind of services to users unobtrusively [11] and in turn the user's interactions with the environment should be as non-intrusive and as transparent as possible [13].

Access to such smart services and devices, such as the fridge and the body sensor, must be controlled by an effective access control system. Access control models have their limitations which hinder their adoption in UbiComp environments. Access control models such as access control matrices [14], rules-based model [15], role-based access control models (RBAC) [16], and Generalised Role-Based Access Control [17] are designed for traditional computing environments. For example, RBAC is a powerful tool for specifying and enforcing an organisational policy in a way that seamlessly maps to an enterprise structure [18]. Moreover, RBAC is considered as a policy natural authorisation approach particularly suited to large-scale distributed environments [19]. However, the traditional access control models are static; they make access control decisions based on the user's immutable attributes (e.g. identities). They are unable to capture the user's or environment's contextual information from the surroundings. Therefore, they can not react to any value changes in the contextual attributes that may have security significance. Such access control models do not suit a dynamic computing environment like UbiComp, where the surrounding context may change

frequently, thus affecting the set of permitted actions of an access requester. Access control in such an environment should be made adaptive in response to those dynamic changes of the users' contextual information which may have security implications. In other words, access control in UbiComp environments should be context-aware.

1.1.2 Context-aware Access Control

A context is defined as "Any information that can be used to characterise the situation of an entity" [20]. It could be related to a user (e.g. user's access location) or a system (e.g. network type) and could be static or dynamic. Static contextual information does not change during the course of a session. Dynamic contextual information changes its values from one session to another, and even during the same session. A UbiComp application has to adapt its services upon the surrounding context; this includes dynamic and static contextual attributes. It should promptly react to the value changes of the contextual attributes. While services in a UbiComp environment are context-aware, a security service that is used to control access to those services should also be context-aware. It should be able to make access control decisions in adaptation to the dynamic changes of the relevant contextual information. For example, if a UbiComp application uses a facial recognition system to authenticate users, it may also be necessary to have an alternative authentication mechanism in place. The latter can be activated, for example, should the lighting system in front of the sensor, that captures the user's facial data, fail. A context-aware access control model incorporates the contextual information in controlling access to sensitive resources. It acquires contextual information from the corresponding context providers then feeds it to the authorisation decision engine that evaluates the contextual information

against the corresponding access control policies (i.e. those written based on the contextual attributes) to produce an access control decision. In a location-aware access control service, for instance, a user's location information is used to control access to a UbiComp service. Any change of the location information may lead to reassessment of the permissions set assigned to the user. Context-aware access control is a powerful tool to enforce context-aware access control policies, thus providing dynamic and smart access control decision-making [21]. In fact, the ability to revise access control decisions for the same access requester, as the surrounding contextual information changes, allows a more fine-grained security service provision in UbiComp environments.

1.2 Research Motivations and Challenges

Enabling access control in UbiComp environments is challenging. For example, consider a new children's hospital that controls access to its resources (e.g. patient records, modality devices, operation rooms, etc) based upon context. Alice and Bob are doctors who run two different clinics in this hospital while subscribing to two different contextual attribute sets. To specify the access rights of Alice over a Magnetic Resonance Imaging (MRI) scanner, the associated access control policy should be aware of Alice's set of contextual attributes, and same for Bob. This means that this access control policy should be expressed in terms of the set of contextual attributes recognised in this application domain. In this hospital there are some children's gift shops scattered throughout. Both hospital and shops share the same Electronic Announcement Board (EAB). The shops use the EAB to publish their offers, whereas the hospital uses it to broadcast any service change and other patient's announcements. The same resource object, EAB, has two different sensitivity levels depending on which system is using it.

The EAB is of a higher sensitivity in the hospital than that in the shop. This is because, from the hospital perspective, it is related to a patient's record that may contain sensitive data; in the shops, however, it is not. The smart system that runs both hospital and shops uses the same contextual information to control access to services in the hospital and shops. However, the way in which the contextual information is represented differs between the hospital and the shops. For example, the hospital may require the location information as a department name, whilst a shop may require it as a distance from the shop location, as there may be multiple shops in the same department. In this case, to specify an access control policy for the EAB, it not only needs to know the required contextual information, but also how this information is represented (i.e. the representation model).

The main challenges in this access control configuration are:

1. How to capture Alice's and Bob's respective dynamic contextual information and feed it, seamlessly, into the authorisation decision engine at runtime.

2. How to accommodate new contextual attributes without imposing considerable modifications to the underlying access control service.

3. How to handle varying sensitivity requirements for a protected resource object.

4. How to deal with contextual information that is expressed using a different representation model.

5. How to minimise the effect of compromising the server-side policy store.

6. How to incorporate the contextual information trustworthiness in such an

access control system. This, in deed, encompass the trustworthiness of the context provider as well as the provided contextual information.

In the above scenario, one could imagine how complex the context-aware access control policies of the resource objects (i.e. the MRI scanner and the EAB) will be under this access control configuration, especially bearing in mind that these policies may need to be uploaded into a device with limited-capabilities as those commonly seen in UbiComp environments. Apart from the complexity, access rights are tightly coupled to the contextual information and its representation model. In these policies, adding new contextual data or remove obsolete one will result in considerable modifications to these access control policies.

1.3 Problem Statement

This project aims to design an adaptive fine-grained access control solution that seamlessly fits into UbiComp environments. The solution should be flexible in supporting the use of different contextual information and efficient, in terms of access delays, in controlling access to resources with divergent levels of sensitivity.

We may be able to achieve this adaptive fine-grained access control by introducing a generic attribute, user's *trust* level or Level of Assurance (*LoA*). We may use this generic attribute to capture and quantify the effect of the user's contextual information on the assurance level in identifying the user. We, then, link this assurance level to the privileges granted to the user. *LoA* is dynamic, since its value changes depending upon the current state of the user's contextual information, thus access control decisions based on *LoA* are dynamic as well.

The above mentioned solution assumes that the contextual information is trusted and of a high quality. In particular, we assume that the contextual information is complete, significant, correct, and up-to-date [22]. In addition, we

assume that the context provider is trustworthy and protected against known attacks (e.g. tamper-proof).

1.4 Aim and Objectives

The aim of this research project is to research, design, implement, and evaluate an access control solution to validate the research hypothesis. To this end, the project objectives are to:

1. investigate the characteristics of UbiComp and specify the requirements for designing an access control solution that will support those characteristics.

2. critically analyse the current access control models (i.e. both conventional and those designed for UbiComp) and evaluate their suitability in line with the identified UbiComp access control requirements. In addition, identify an appropriate method of achieving access control in this dynamic environment;

3. design an access control solution that seamlessly accommodates any set of contextual information and adaptively controls access to UbiComp services.

   will satisfy the UbiComp access control requirements;

4. evaluate the designed solution and validate if the research hypothesis is true. If it is not, to explore the reasons for this;

5. publish the research results; and

6. write the PhD thesis.

1.5 Research Achievements and Publications

The following lists the achievements of the research presented in this thesis: -

1. The Context-Risk-Aware Access Control model (CRAAC)

   CRAAC is an adaptive LoA-linked access control model. It controls access to resources with varying levels of sensitivity based upon the state of an access requester's contextual information. It supports adaptive access control decisions, since an access control decision for the same access requester on the same resource object may vary each time. The variation depends on the level of assurance of the access requester, which is based upon the access requester's current contextual information.

   - CRAAC supports fined-grained access control, since it, virtually, accommodates any set of contextual attributes. This level of abstraction is achieved by the use of a trust-related parameter (i.e. Requester's Level of Assurance (RLoA)). In addition, in controlling access to a resource object, CRAAC accommodates the resource object's sensitivity level, that may be dynamic as well, to provide a more fine-grained access control.
   - CRAAC is flexible; adding new contextual attributes or removing obsolete ones will not significantly affect the underlying access control system.
   - CRAAC supports four modes of working to accommodate different access control requirements.

2. Methods for context to *LoA* conversion

   - LoA-to-Weight Conversion (*L2WC*). Converting the individual contextual attributes to the corresponding levels of assurance using Rank

Order Centroids (*ROCs*).

- RLoA Aggregation. Aggregating the user's multiple attributes' LoA values into an aggregated level of assurance. Two types of relationships amongst contextual attributes are identified and used for the aggregation (i.e. *Weakest-Link* and *Elevating*).

3. An access control architecture along with its components to support the novel CRAAC services and the CRAAC four modes of working.

4. Evaluation

   - Performance evaluation of the CRAAC model.
   - Denial of service attack investigation.

The publications produced in this research are:

1. Ali Ahmed & Ning Zhang, "An Access Control Architecture for Context-Risk-Aware Access Control: Architectural Design and Performance Evaluation", accepted to be appeared in SECURWARE 2010.

2. Ali Ahmed & Ning Zhang, "Towards the realisation of context-risk-aware access control in pervasive computing". Telecommunication Systems, December, 2009.

3. Ali Ahmed & Ning Zhang, "CRAAC: Context-Risk-Aware Access Control", Informal Workshop on Formal Approaches to Ubiquitous Systems, Imperial College, 14-15 September 2009.

4. Ali Ahmed & Ning Zhang, "A Context-Risk-Aware Access Control model for Ubiquitous environments", IMCSIT' 2008: International Multiconference on Computer Science and Information Technology., October, Pages 775-782, 2008.

1.6 Thesis Structure

This thesis investigates the contextual attributes that may influence the assurance level of an access requester in UbiComp environments. It defines those attributes and suggests the corresponding LoA values. It also proposes two methods that may be used to aggregate the contextual attributes LoA values into one access requester's LoA value. The use of an aggregation method depends on the relationship among the contextual attributes used. It then proposes an access control model, namely Context-Risk-Aware Access Control (CRAAC) for UbiComp environments. CRAAC has four modes of working each of which supports a different access control configuration/requirement. The thesis also proposes an architecture that realises the CRAAC vision and supports the CRAAC four modes of working. A prototype of the CRAAC system has been developed and the performance of the model has been investigated using the prototype. The security of the system has been also analysed in terms of safety and denial of service attack.

In detail, Chapter 2 surveys the traditional access control models. Chapter 3 discusses and evaluates the access control models proposed for UbiComp environments. It also identifies the shortcomings of those models and suggests the best way forward. Chapter 4 presents the CRAAC model. It discusses the basic model building blocks such as the contextual attributes identification, level of assurance derivation, methods to derive an aggregate access requester's level of assurance, and CRAAC four modes of working. Chapter 6 proposes the CRAAC architecture, along with its components, that realises the CRAAC services. It describes the architectural components in detail. It also shows how the RLoA-only working mode uses the architecture. Then, it evaluates the RLoA-only mode performance and security. Chapter 7 shows how the AttributeLoA-only mode works and which architectural components will be used. It discusses the mode

potentials and concerns compared against the RLoA-only mode. It then investigates the performance of the AttributeLoA-only mode. Chapter 8 concludes the thesis and suggests directions for future work.

Chapter 2

Traditional Access Control Models

2.1 Chapter Introduction

This chapter surveys some access control models that are proposed for traditional computing environments. It discusses two groups of these legacy models, Discretionary Access Control (DAC) and Mandatory Access Control (MAC). As an example of the DAC model, the chapter presents the Access Control Matrix (ACM). It also briefly highlights various implementations of the ACM model such as the Access Control Lists (ACL), capabilities and authorisation relations. This chapter mainly focuses on discussion about the Role-Based Access Control (RBAC) model. RBAC is the basis of a new generation of access control solutions proposed for UbiComp environments. These solutions will be covered in Chapter 3, which critically analyses those solutions and evaluates their suitability. In conducting the survey, the good design principles of the models are examined for possible inclusion in our proposed model.

The organisation of the chapter is as follows:

- Section 2.3 discusses the DAC model using the ACM with its different implementations (i.e. ACL, capabilities, and authorisation relations).
- Section 2.4 introduces the MAC model.
- Section 2.5 describes the RBAC core model, role hierarchy, constraints, and model restrictions.
- Finally, Section 2.6 summarises the chapter.

2.2 Basic concepts

In this section, the basic concepts and terms used in access control in computer systems are introduced. The term access control is used to restrict the actions a legitimate entity can perform on a given resource object [23]. It enables an authority to control access to a certain resource object by defining an associated policy (i.e. access control policy). The policy contains all permissible actions that an entity can initiate on the given resource object. A typical access control system consists of a *subject*, an *object*, a *permission*, and *credentials*. A *subject* is an entity that seeks access to a resource object and is sometimes referred to as an access requester. This may be a human user, a software application or a hardware device. An *object* is actually a target protected by the access control system. A *permission* is an access right for a *subject* over an *object*. It corresponds to a privilege that a *subject* owns over a certain *object*. A *subject* uses some *credentials* to gain access over a particular *object*. A *credential* is defined as "a piece of information that is used to prove the identity of a subject" [24]; passwords, crypto keys, and biometric data are examples of such credentials.

2.3 Discretionary Access Control (DAC)

Discretionary access control (DAC) is a class of access control models that controls access to an object based on the identity of a subject [25]. The identity could be the subject's user name or the subject's group membership. The word *"discretionary"* means that a subject that *owns* a certain access right over an object can pass the access right to other subjects on his/her discretion. In other words, DAC allows an owner of a particular access right to a specific object to pass on the access right to other subject(s) based on the owner's personal preferences. This capability makes the DAC model flexible in supporting commercial solutions where no strict information flow is required. For example, manipulating a shared folder on a server can benefit from this capability in DAC. The owner (e.g. Bob) of an access right (i.e. *read*) over the shared folder can easily pass on this access right to another subject (e.g. Eve) by creating a user-name/password pair associated with Eve. In fact, no real control on the flow of information is provided in DAC. Since Bob can pass his *read* right on to any body at his discretion, the system manager, for instance, is unable to control this. Thus, the manager can not ensure the flow of information in the underlying system. This property in DAC (i.e. lack of information flow control), actually, increases the possibility of unauthorised access [23]. Therefore, DAC is not suitable for military applications that require a rigorous control of information flow.

ACM is, perhaps, one of the first discretionary access control models for computer systems. It defines the access rights of each subject over a set of resource objects managed by the system [23]. In this model, a matrix is constructed in which there is a column for every resource object and a row for each subject. Therefore, a cell in this matrix specifies the access rights of a certain subject over a particular object. For example, Table 2.1 shows a typical ACM for a system

that manages a printer, a photocopier, and a seminar room. The subjects of this system are Alice, Bob, and Eve. The table shows, for example, that the access rights of Bob over the photocopier are 'copy' and 'scan', whereas Eve can only use it as a 'fax'. There are various implementations of the ACM. The well-known ones are ACL, capabilities, and authorisation relations.

Table 2.1: Access Control Matrix Sample

Subject	Printer	Photocopier	Seminar Room
Alice	*print*	*copy* *scan* *fax*	*order equipment* *access* *change PIN*
Bob	*order parts*	*copy* *scan*	*access*
Eve	*print*	*fax*	*access*

2.3.1 Access Control Lists

ACL [26] is, perhaps, the most popular implementation of the ACM. ACL can be represented as storing the ACM in a columnar way. In other words, each resource object will have an associated list that defines, for each subject, the set of legitimate actions the subject can perform on it. A sample ACL is shown in Figure 2.1 that describes which subject can perform what action(s) on the photocopier. The ACL implementation is object-centric, since it specifies an object's legitimate access modes. Thus, it is straightforward to update an object's access modes by modifying the associated ACL [23]. It is also easy to revoke an object's access modes by replacing the existing ACL with an empty one. However, determining the access rights of a subject is not easy. It requires every ACL in the system to be checked against the subject.

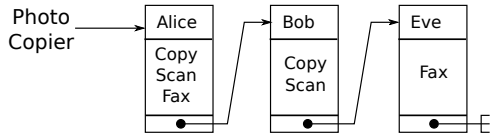


Figure 2.1: The Access Control List for a photocopier Machine

2.3.2 Capabilities

Capabilities are another implementation of the ACM. Unlike the ACL approach, capabilities store the ACM in rows. In other words, each subject in the system will have an associated capability over a set of managed resource objects. Capabilities are subject-centric as depicted in Figure 2.2. This actually solves the problem of determining the set of allowed actions by a specific subject by just examining the subject's associated capabilities. However, revoking an object's access modes requires all the capabilities in the system to be examined.

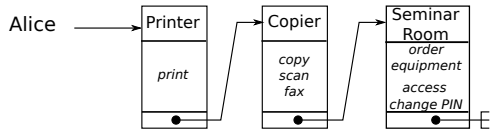


Figure 2.2: Alice's Capability

2.3.3 Authorisation relations

An authorisation relation is another implementation of the ACM that is inspired from the relational databases. As shown in Table 2.2, a relational table is created where each row represents an access operation a subject can perform over a specific object. This implementation does not actually favour one preview over another, as ACL and capabilities do. It is easy to get a specific subject's access rights by sorting the table by subject, which actually corresponds to the capability method. Similarly, sorting this authorisation table by object produces the same effect as the ACL method.

Table 2.2: Authorisation Relations

Subject	Access Right	Object
Alice	*print*	Printer
Alice	*copy*	Copier
Alice	*scan*	Copier
Alice	*fax*	Copier
Alice	*order equipment*	Seminar Room
Alice	*access*	Seminar Room
Alice	*change PIN*	Seminar Room
Bob	*order parts*	Printer
Bob	*copy*	Copier
Bob	*scan*	Copier
Bob	*access*	Seminar Room
Eve	*print*	Printer
Bob	*fax*	Copier
Bob	*access*	Seminar Room

2.3.4 DAC Summary remarks

It is worth noting that managing the ACM in a large-scale distributed environment is troublesome. ACM is a static access control solution. In this model, subjects and objects need to be pre-defined. In addition, the access control decisions are immutable. Additional constraints can not be imposed easily. Therefore, DAC is not suitable for the UbiComp environments that require access control to be adaptive to some dynamic constraints (i.e. contextual information). Moreover, DAC can not cope with information leakage caused by the weak control of information flow. MAC was, in fact, proposed to overcome this weakness in DAC.

2.4 Mandatory Access Control (MAC)

Unlike DAC where access rights are defined by the resource owner based on the resource owner's discretion, mandatory access control enables the access rights to

be determined by a manager or a central authority of the system. In MAC, each resource object in the system is labelled with a sensitivity level and each subject is assigned a clearance level. This process is performed by a central authority not by the resource owner. Access to a resource object is restricted to those subjects who possess a valid clearance level (i.e. authorisation) [25]. MAC defines a meticulous flow of information dissemination; thus the likelihood of potential illegitimate access is reduced compared to the DAC model. The idea of labelling an object with a sensitivity level, which a subject has to satisfy in order to gain access to this object, is interesting. This is one of the principles used in our research that will be discussed in detail in Chapter 4. It is worth emphasising that MAC suits access control requirements where an object's access right is determined by a central authority and not at the discretion of the owner of the resource object. Thus, it is suitable for governmental and military applications. However, MAC is too rigorous for, for instance, commercial domains. A more flexible model that can suit such domains is needed. The model was proposed as role-based access control.

Unlike DAC, MAC model is vulnerable to the convert channel attack. Lampson [27] defined the convert channel as "(channels) not intended for information transfer at all, such as the service program's effect on system load". The idea of the attack is centred on adding capabilities to transfer information between entities, which are not supposed to do so. In other words, it is a kind of a collusion between the sender and the receiver in a clear violation of the MAC security policy [28]. Any MAC model should pay attention to such a vulnerability, although it is hard to detect.

2.5 Role-Based Access Control (RBAC)

RBAC is an access control model that governs access to a resource object based on a subject's organisational role. A role represents certain activities a member of staff (i.e. subject) can perform as a part of his/her organisational responsibilities. Instead of assigning access rights to subjects directly, RBAC assigns access rights to roles and then maps the roles to subjects. This facilitates scalable and efficient management of an individual subject's access rights, since no access rights are accorded to subjects directly. This is, in fact, true if the number of roles managed by an organisation is less than the number of subjects. Adding a new subject or changing the responsibilities of an existing one would be a simple task, which only requires the addition or modification of the subject-role mappings. In other words, the access control administration becomes simple as a result of using roles [16].

In RBAC, constraints can be imposed on the access requests to prevent unauthorised access or malicious activities. RBAC supports the *"least privilege"* security principle, in which a subject is given the least privilege which sufficiently allows the subject to perform the task in hand (i.e. current role) [23]. In this way, RBAC prevents the leakage of access rights to unauthorised entities and reduces the risk of fraud. This also ensures *integrity*; nobody is allowed to modify a data item without a permission. The NIST standard [16] describes a family of services or components an RBAC system should provide. These components include Core RBAC, Hierarchical RBAC, and Constrained RBAC. Since the research carried out in this thesis is based on the RBAC model, an overview of these components will follow. The overview focuses on the aspects of RBAC that could be of interest in our problem domain (i.e. context-aware access control for UbiComp environments).

2.5.1 Core RBAC

The Core RBAC is the fundamental component in any RBAC-based system. It includes the minimum set of functions that are needed to realise the RBAC model. Two essential functions are introduced by the Core RBAC: User Assignment (UA) and Permission Assignment (PA). UA defines a set of roles to which a user can be mapped. For example, a user (e.g. Bob) could be assigned to the role set of {*Lecturer, Student Mentor, Admission Deputy*}. In fact, UA determines the super set of roles to which a user can be mapped as a part of the user's organisational duties. As seen in the example, the user-role relationship is many-to-many. A user may be holding more than one role and a role may be held by more than one user. PA, on the other hand, specifies the set of allowed actions for a given role. It corresponds to assigning access rights to roles. For instance, the *Lecturer* role could claim the following access right set {*access seminar room, use projector, use white board, etc*}. The role-permissions relationship is many-to-many in the same sense as in UA.

User *sessions* are an important element in the Core RBAC. A user's session specifies a time window that allows a role to be activated and deactivated for this particular user. In other words, a user's session contains the active roles of the user. The set of allowed actions a user can perform is determined by the active roles of the user, as the access rights are actually released at run-time to those active roles. In other words, the combination of all the permissions assigned to the active roles of a user constitutes the user's access right set in a session. The relationships between user, roles, permissions, and sessions are depicted in Figure 2.3. The relationship between users and sessions is one-to-many. A user may have many sessions, but a session is dedicated to exactly one user. More advanced RBAC-based systems can be built on top of the Core RBAC. They

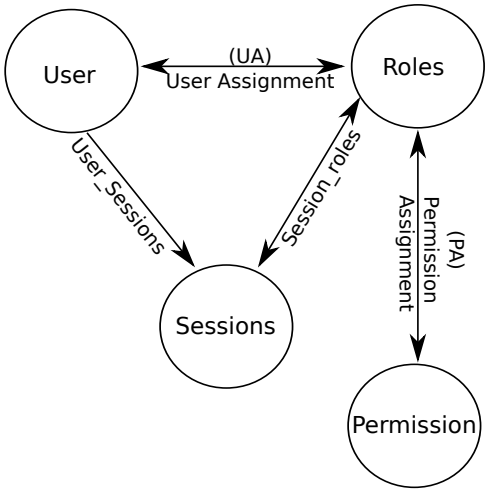


Figure 2.3: The Core RBAC

extend the Core RBAC to support, for example, role hierarchy or constrained access control.

2.5.2 Role Hierarchy

Many organisations naturally structure their roles hierarchically to benefit from inheritance among roles. In fact, the role hierarchy facilitates the administration of organisational roles. This is a well-known generalisation-specialisation pattern. For example, a specialised electrical engineer may also inherit the access rights of a more general role (e.g. engineer). RBAC utilises a model component (i.e. Hierarchical RBAC) to deal with the role hierarchy. Figure 2.4 shows an RBAC system with role hierarchy. Generally, multiple inheritance is allowed in hierarchies unless restricted. That is, a role may be composed of several other roles. A role that inherits from multiple roles claims all access rights of the inherited roles plus its own access rights.

An example of multiple role inheritance is given in Figure 2.5. An *"Engineer Manager"* role may inherit the access rights of both *"Engineer"* and *"Manager"*

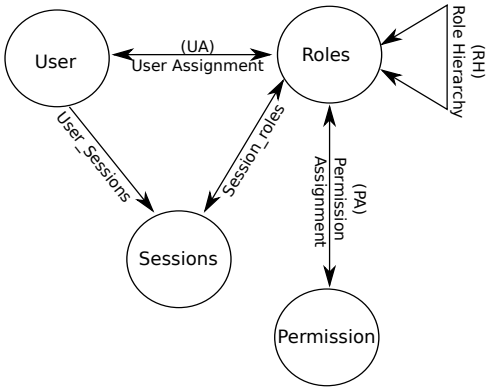


Figure 2.4: The Hierarchical RBAC

roles. The Hierarchical RBAC, which supports access rights inheritance, provides flexibility in roles and access control management. It is possible to apply additional constraints to restrict such role inheritance. For example, some roles are mutually disjoint. A clerk at a bank counter, who can release funds, can not be the one who verifies a client's signature. An organisational role hierarchy must not allow such roles to inherit from one another. In fact, there is another model component, called Constrained RBAC, that is especially proposed for this purpose.

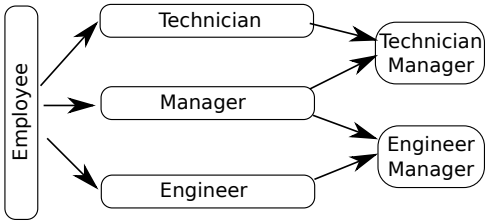


Figure 2.5: Example of a Role Hierarchy

2.5.3 Constrained RBAC

Constraints are necessary to ensure that no subject is given sufficient privileges to misuse the system [23]. In other words, constraints are imperative in order to support the *"least privilege"* principle. The Separation of Duties constraint (SoD)

is the common constraint used in Constrained RBAC. The purpose of SoD, as stated by Gligor et al. [29], is "to ensure that failures of omission or commission within an organisation are caused only by collusion among individuals and, therefore, are riskier and less likely, and that the chances of collusion are minimised by assigning individuals with different skills or divergent interests to separate tasks". Put simply, in any organisation, an entity which authorises an activity can not be the same entity that carries out the activity. The Constrained RBAC described in [16] recognises two types of SoD: Static SoD (SSD) and Dynamic SoD (DSD). Gligor et al. [29] has defined four other types of SoD policies and linked these types to the RBAC model. Figure 2.6 shows how constraints are used in RBAC. As depicted in the figure, constraints can be used to restrict user-role and role-permission assignments, role hierarchy, and user-sessions.

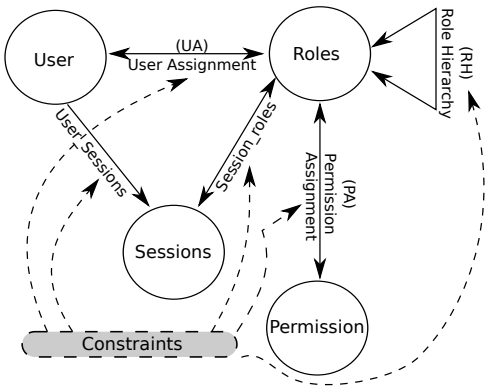


Figure 2.6: RBAC with Constraints

2.5.3.1 Static Separation of Duty

SSD solves the conflict of interest problem. A particular role holder (i.e. subject) must not gain the access rights of a mutual disjoint role holder. SSD actually enforces certain requirements on the UA function. In other words, SSD places constraints on the role membership. Thus, a user has to satisfy certain requirements before being assigned to a specific role. That is, if a user is granted a

specific role, the user can not be assigned another role that is mutually disjoint with the first role. As a consequence, the number of available permissions a user can have is restricted or reduced. Thus, SSD controls, indirectly, the user's permission space [16]. It draws the boundaries around the permissible set of actions a user can do as a part of the user's organisational duties.

2.5.3.2 Dynamic Separation of Duty

Like SSD, DSD tries to restrict the permissions a user can have. Nevertheless, DSD imposes certain constraints on the available permissions in the user's permission space. It is worth emphasising that the permissions associated with a certain role are not released to a user unless the user has activated that role. And this is when DSD is applied. In other words, DSD imposes constraints on the role activation process. This supports the *"least privilege"* security principle, discussed earlier, in a way that a user may have different security levels depending on the task in hand and permissions are not granted unless a role is activated. This kind of constraint is applied at access-time, i.e. when a user activates a role. A good example to show the difference between SSD and DSD is given in [30]: "A static policy could require that no individual who can serve as payment initiator could also serve as payment authorizer. This could be implemented by ensuring that no one who can perform the initiator role could also perform the authorizer role. Such a policy may be too rigid for commercial use, making the cost of security greater than the loss that might be expected without the security. More flexibility could be allowed by a dynamic policy that allows the same individual to take on both initiator and authorizer roles, with the exception that no one could authorize payments that he or she had initiated".

2.5.4 Advantages of RBAC over DAC/MAC

Rather than associating access rights to individual objects as in, for instance, ACM, RBAC assigns access rights to certain roles. This maps naturally to an organisational structure. RBAC is considered a more generalised form of both DAC and MAC. DAC and MAC could be seen as special cases of RBAC [31] or, in other words, examples of policy configurations inside RBAC. RBAC is a powerful model to specify and enforce organisational policies in a way that seamlessly maps to an enterprise structure [18]. RBAC is considered as a policy natural authorisation approach particularly suited to large-scale distributed environments [32].

2.5.5 What is Missing in RBAC?

Despite of a number of advantages, RBAC still has some limitations. Three of these limitations have been identified by Sejong Oh [33]:

1. Roles inheritance in RBAC does not fully reflect the same process in real organisations. For example, if a role R_1 (i.e. a higher role) inherits from another role R_2 (i.e. a lower role) this means that R_1 inherits the full permissions set of R_2, whereas in real life organisations, a higher role only inherits a partial permission set of a lower role.

2. No clear separation between the "task" and "role" concepts. In real organisations, sometimes a task may require the involvement of multiple roles.

In addition, in emergent computing environments such as UbiComp and GRID, access control decisions are dynamic, as the privileges and capabilities of users may change. For example, access rights in a UbiComp environment may not depend solely on the users' identities [34]. They may also depend on the context in which an access request is made. This includes the user's context such as the

access location and access time, and the system's context such as the system load and network state. The RBAC model, in fact, is not able to consider the contextual information in access control decision-making. It can not capture the contextual information from an access requester's surrounding environment nor can it adapt its access control decision in response to any change of the contextual information. The fact that access control policies in RBAC are presumably static (i.e. they follow the same access control requirements regardless of any change in the surrounding environment) hinders the application of RBAC to achieve a more fine-grained access control required in UbiComp environments. There is a need for a new access control model that can overcome the limitations mentioned above. In other words, a model that is neither subject-centric nor assumes the use of static policies like RBAC. Therefore, a new generation of access control models has been proposed under the name of context-aware access control models. This generation of access control models encompasses many access control solutions that share the same concept of using contextual information in controlling access to resource objects.

2.6 Chapter Summary

This chapter has provided an overview of the basic concepts of access control as well as some of the well-known traditional access control models such as DAC, MAC, and RBAC. DAC is not suitable for UbiComp environments, since it is a static model that lacks a proper control of information flow. An owner of a particular access right can pass the access right on to any other subject without restrictions (i.e. at the owner's discretion). However, DAC is suitable for commercial applications, where such a limitation may be tolerated. MAC provides

a rigorous control of information flow that is required in military and governmental applications. Access control policies in DAC and MAC are immutable. Access control decision-making follows static access control requirements that do not exist in UbiComp. UbiComp requires a dynamic and flexible access control solution that adaptively adjusts access control decisions based on the surrounding environment (i.e. context). The RBAC model was introduced to overcome the current limitations of DAC and MAC. RBAC is a flexible model that is a more generalised form of DAC and MAC. RBAC-based solutions range from simple to more sophisticated access control systems. This, in fact, depends on which RBAC model components are used (e.g. Core RBAC, Hierarchical RBAC, etc). Although RBAC has advanced the research in the area of access control, it cannot cope with the fundamental UbiComp requirement of accommodating contextual information in access control decision-making. RBAC is the basis of a new generation of access control models. Those models extend RBAC to accommodate the contextual information in access control decision-making. The following chapter describes some of the most well-known context-aware access control models.

Chapter 3

UbiComp Access Control: A Survey

3.1 Chapter Introduction

This chapter discusses some of the existing access models that have been proposed for UbiComp environments. This family of access control models is largely context-aware. Thus, they are collectively called the Context-Aware Access Control (CAAC) model. CAAC, typically, extends the traditional RBAC model, discussed in Chapter 2, to accommodate the contextual information in controlling access to sensitive resource objects. This chapter highlights some recent streams in access control in UbiComp environments. In surveying those solutions, the discussion mainly focuses on the strengths and weaknesses of the architectural components of the proposed access control models. The good design principles, in those models, are emphasised to be used in our proposed model. Finally, this chapter outlines the best way, from the author's point of view, to advance the research in access control for UbiComp environments, in order to overcome the

identified weaknesses of the existing access control models for UbiComp environments.

The remaining part of this chapter is structured as follows: -

- Section 3.2 introduces the CAAC family and highlights some CAAC-based proposals.
- Section 3.3 discusses some related efforts in advancing access control in UbiComp environments.
- It also introduces the effort to consider a user's trust in access control decision-making.
- Section 3.4 critically analyses the CAAC-based models.
- It describes the approach that the CAAC model uses to accommodate the contextual information in access control decision-making.
- Section 3.5 introduces our proposal to advance the research in access control for UbiComp environments.
- Finally, Section 3.6 summarises the chapter.

3.2 Context-Aware Access Control (CAAC) Models

The CAAC model denotes a family of access control proposals that control access to resources by using the user's contextual information. A CAAC-based model uses the contextual information as additional [1] constraints to govern access to sensitive resources. It is worth emphasising that most CAAC-based solutions are

[1] Additional to the SoD constraints discussed earlier in chapter 2

built on the RBAC model discussed earlier in Chapter 2. As shown in Figure 3.1, the CAAC model introduces a new type of constraints, called contextual constraints, to govern the UA and PA functions. A user is granted a certain role iff the user satisfies a particular contextual constraint (i.e. specified in the UA policy). Similarly, a permission is released to a given role, provided that the role holder satisfies a certain contextual constraint (i.e. specified in the PA policy). Many access control proposals are CAAC-based. A detailed discussion of some of those proposals will be given in the following subsections.

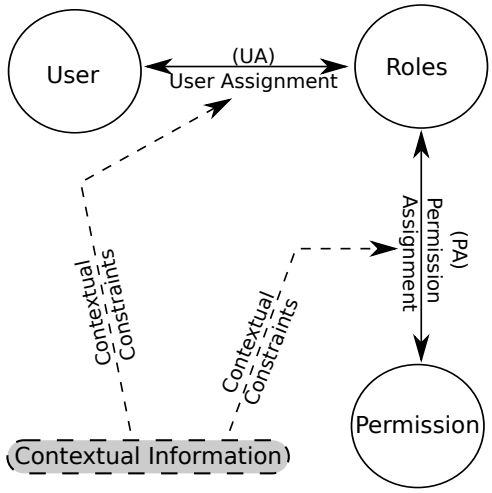


Figure 3.1: CAAC Conceptual View

3.2.1 Generalised Role-Based Access Control

The Generalised Role-Based Access Control (GRBAC) is, perhaps, the earliest CAAC-based proposal. It was proposed by Covington et al in [35]. The proposal was motivated by and designed for *smart home* environments. GRBAC extends the concept of *role* to encompass two new types of roles: *environment role* and *object role*. An *environment role* captures an environment state, which represents a system's contextual information that should be incorporated into access control decision-making. On the other hand, an *object role* is used to capture an object's

sensitivity level. For example, an object may have, depending on the situation, a confidential role or top classified role. Thus, for a subject (Sub) to perform a certain task (T) over an object (Obj) in an environment (Env), it requires the following: -

1. Determining the object role $Role_{Obj}$.
2. Determining the subject role(s) $Role_{Sub}$ (i.e. the traditional RBAC role).
3. Determining the environment role(s) $Role_{Env}$.
4. Checking if T is allowed for $Role_{Sub}$ on $Role_{Obj}$ when $Role_{Env}$ is active.

Using the extended roles(i.e. *object role* and *environment role*) removes the subject-centric limitation of the RBAC model. An access control policy could now be written from an object's perspective, environment perspective, or any possible combination of both [35]. However, the access control decision-making in GRBAC is more complex than that of RBAC. It requires the use of a more complex system architecture than the traditional RBAC model. This is due to the introduction of the new extended roles [35, 36].

3.2.2 Spatio-Temporal Models

Many access control solutions proposed in literature use the location and time contextual information to control access in UbiComp environments. Examples include the Temporal RBAC (TRBAC) model [37], the Generalized Temporal RBAC (GTRBAC) [38], and the Spatial RBAC (SRBAC) [39]. In these solutions, roles/permissions are granted to a user in specific time intervals and/or if the user is within a particular location.

The TRBAC model extends the traditional RBAC model by introducing a temporal constraint into the access control policy specification. The inclusion of

the temporal constraint provides a mechanism to enforce time-dependent access control policies [2]. Thus, a role is enabled or disabled iff a certain temporal constraint is satisfied. The GTRBAC model further extends the TRBAC model by introducing the notion of an *activated role*. In other words, GTRBAC differentiates between role enabling and role activation. An enabled role indicates the possibility of a subject to claim it along with its permission set; however this is not done yet. An active role, on the other hand, is an enabled role that has been activated by at least one subject in a session. This means that a subject has acquired the permission set of that role. The notion of the active role helps to determine the currently running roles. This can be used to monitor the resource usage and activities taken place. GTRBAC imposes several temporal constraints on the role activation, enabling times, and UA/PA policy specification [38].

The SRBAC introduces a location-dependent constraint. It restricts the set of permissible actions a subject can perform based on the subject's location information. The location space is divided into multiple zones. An access permission is granted if the role condition is satisfied and the subject is within a specific zone. Table 3.1 shows an example of how permissions are assigned to roles based on the location information. It depicts a permissions list associated with a *customer_role*. For example, a *customer_role* holder in $zone_1$ has a permission set of $\{p_1, p_2, p_3\}$, whereas in $zone_2$, the permission set is $\{p_4\}$. The SRBAC model, as observed by Zhang et al [40], suffers from a lack of a semantic meaning of the position information. It also does not support the use of geometrically bounded roles.

Table 3.1: Location Permission Assignment List in SRBAC

Roles	Location	Permissions
customer_role	Zone_1	p_1, p_2, p_3
customer_role	Zone_2	p_4
customer_role	Zone_3	$\emptyset$

There are a number of other proposals that fall into this spatio-temporal access control approach including those described in [41, 42, 43, 44, 45, 46, 40, 47, 48, 49, 50]. Those proposals support either location-aware, time-aware, or both of them.

3.2.3 Dynamic Role-Based Access Control

A common feature in the proposals discussed above is that they take the contextual information into account at the beginning of an access session. In other words, contextual information is evaluated when an access control request is received. Thus, when an access control decision is made, no further evaluation for contextual information is performed. They make no effort in adjusting the access control decision during the course of an access session. In addition, models such as spatio-temporal are restricted to only time and location information. Contextual information is not just location and time. A fine-grained access control in UbiComp environments should incorporate every relevant contextual information that has significance on access control.

To bridge this gap, the Dynamic Role-Based Access Control (DRBAC) model was proposed [51]. DRBAC dynamically adjusts the permission assignment as well as the role assignment based on a subject's contextual information. The contextual information could be any piece of information not just time or location. It uses a Central Authority (CA) to manage the role hierarchy and to grant roles to users. To perform this task, the CA dispatches an agent to a user's device for every role the user has been assigned to. The agent monitors the user's context. It changes the active role dynamically based on the changes in the contextual information. It uses a state machine to express the user's roles. Typically, the user is assigned an initial role and any change in the contextual information will

be detected by the agent, which will, in turn, trigger an event. The event will result in a transition between the current state (i.e. current role) to the next state (i.e. new role) in the state machine. In this way, the DRBAC model changes the user's role dynamically in response to the user's contextual information changes, thus achieving adaptive context-aware access control.

DRBAC suffers from a number of drawbacks, some of which have been highlighted by the authors themselves. For example, implementing the model can increase the complexity of the applications concerned. Each role requires a role state machine running on the user's device. As the number of roles supported in the system increases, the complexity of the system is also increased. This is particularly troublesome for the resource-restricted devices that are commonly used in UbiComp environments. In addition, as mentioned in [42], the paper has not illustrated how the DRBAC model may be applied practically in an application. In spite of these drawbacks, DRBAC is considered one of the most pioneering access control solutions for UbiComp environments.

3.2.4 CAAC-based Models with Architectural Components

To support the CAAC approach, there is a need for an architecture that could feed contextual information into access control decision-making. In other words, a CAAC architecture should encompass two main functional blocks (i.e. infrastructures). One block would be for the contextual information management, and the other block would be for access control. However, the traditional access control architectures only accommodate access control, thus not appropriate for supporting context-aware access control. To overcome this limitation, there have been increased efforts to design architectures that support context-aware access control. The following gives a brief survey of some of those efforts.

3.2.4.1 UbiCOSM

UbiCOSM [5] is another CAAC-based model that evaluates the current context state of a mobile user in real-time. It uses the context state to control access to protected resources. As seen in Figure 3.2, UbiCOSM is a modular middle-ware. It uses external architectural modules such as CARMEN [52] for low-level entity identification and context management. However, the tight-coupling between access permissions and contextual information in UbiCOSM makes it difficult to accommodate different access control requirements, policies, and application domains.

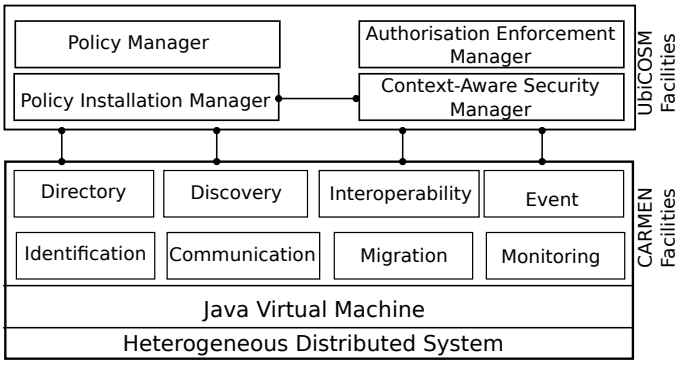


Figure 3.2: UbiCOSM Middleware Services [5]

UbiCOSM separates the authorisation engine from the access control enforcement by introducing the Authorisation Enforcement Manager. This provides high level of modularity. Since the access control enforcement is application-dependent, one can replace an existing enforcement point with another depending on the application domain and requirements.

3.2.4.2 OpenAmbient

OpenAmbient [6] presents a web service-based architecture that preserves privacy and protects resources in ambient environments. The architecture, as depicted in Figure 3.3, uses the ContextToolkit [53] for contextual information gathering. The

ContextToolkit acts as an ambient information provider that resides internally in the OpenAmbient architecture. Similar to UbiCOSM, OpenAmbient has a high degree of modularity, allowing the use of external modules (e.g. ContextToolkit). The paper, however, only describes the basic OpenAmbient architecture and does not explain how the architectural components interact and communicate with each other to reach an access control decision. OpenAmbient also follows the same UbiCOSM convention of separating the access control enforcement and the authorisation engine. It supports the former by introducing the Enforcer, while the Evaluator is meant for the latter function.

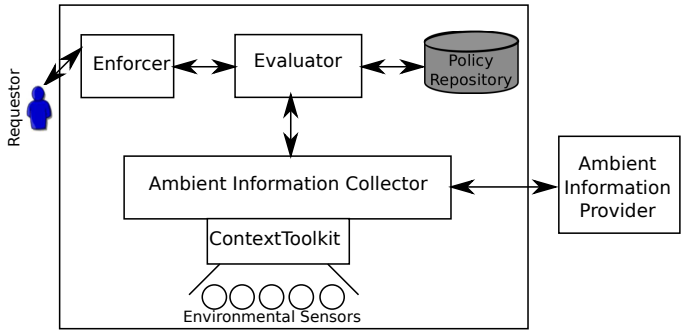


Figure 3.3: OpenAmbient Architecture [6]

3.2.4.3 The Gaia Architecture

Gaia [7] is one of the earliest efforts to develop context-aware applications. Gaia is regarded as a meta operating system, which provides an environment for the development and execution of active spaces. The main contribution of Gaia is the proposal of the first meta operating system that has context-awareness as a built-in service. In other words, context-awareness was a fundamental requirement in the design of the operating system, rather than as an add-on service to an existing operating system. Moreover, Gaia shifts the task of context management from the application layer to the operating system layer. Thus, when developing an

active space application, one does not have to worry about the task of managing contextual information. However, the Gaia architecture, as depicted in Figure 3.4, does not address access control specifically. Therefore, Gaia does not propose any building blocks for access control. The lack of a standard access control infrastructure in Gaia makes it hard for different systems to cooperate seamlessly. This is because a system may have its own terminologies and architectures.

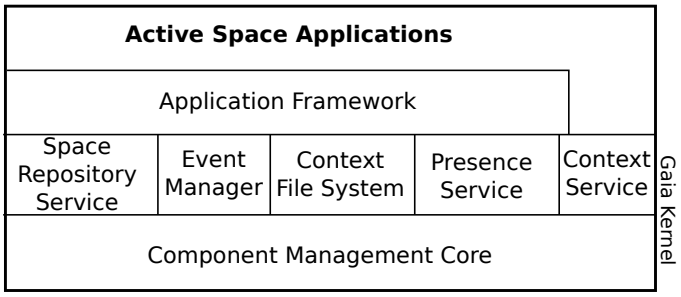


Figure 3.4: The Gaia Architecture [7]

To bridge this gap, the work in [54] is proposed. It describes an access control system for the Gaia Active Spaces system. Its architecture shows a set of applications that run on behalf of users to access certain devices, where software services act as interfaces for the applications to access devices. The access control service consists of an Interceptor module and Access Control Policies. The Interceptor intercepts all access requests, evaluates these requests, and only allows authorised requests.

3.2.4.4 Context-Constrained Access Control (CoCoA)

The CoCoA model described in [8] is a context-constrained authorisation framework designed for GRID environments. It is built on the GT4 authorisation framework [55] by adding additional modules to support context-awareness. These modules, as seen in Figure 3.5, are Context Authority, Context PIP, Context PDP, Context Session Service, and Notification API. The Context Authority provides

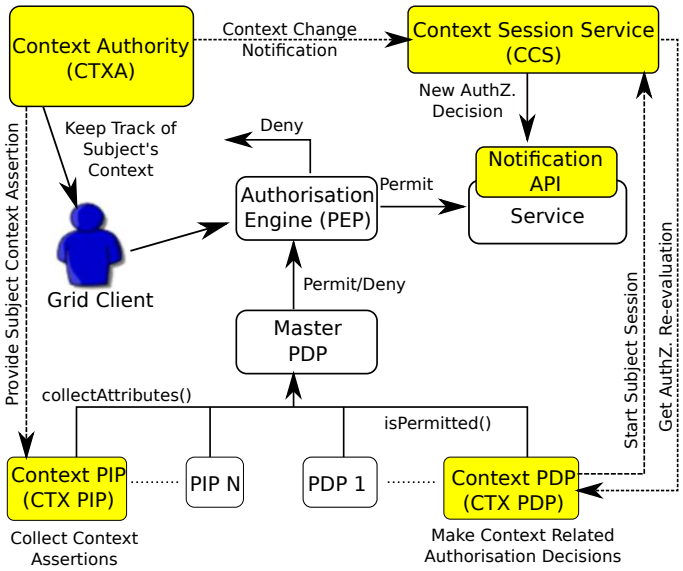


Figure 3.5: The CoCoA Architecture [8]

real-time, up-to-date contextual information. In addition, it may carry out further operations to interpret some specific contextual information. The Context PIP provides an assertion of a subject's contextual information, supplied by the Context Authority to the Context PDP, before the latter makes a context-aware access control decision. The Notification API allows a GRID service provider to be notified whenever a renew authorisation decision is available for reinforcement. Upon the receipt of a context change notification, the service provider may terminate the user's access session. The CoCoA design has adopted a number of good practices such as reducing the level of coupling among architectural components and compliance to international standards (i.e. the GT4 authorisation framework standard). However, owing to different motivations, CoCoA is not readily applicable to the context that we are examining. CoCoA is designed for GRID applications, thus one of the assumptions used in its design is that an access session is typically for a job execution, which may last for hours or even days. Therefore, the CoCoA model monitors a user's contextual information during the course of an access session. Should there be any change in the user's context, it

terminates the user's access session. This is in stark contrast to the problem we are addressing in this thesis. This thesis concerns data accesses in a UbiComp environment where an access session is typically short (i.e. for one data item access only). Furthermore, we would like to quantify the impact of contextual information on the level of assurance associated with the data access. We would then feed this level of assurance into the access control decision-making at the beginning of an access session and monitor its change during the access session.

3.3 Recent Proposals

3.3.1 Generalised Context-Based Access Control

Filho and Martin in [9] have proposed a generalised context-based access control model. In this model, access to resource objects is controlled solely based on contextual information. The model is designed for an open UbiComp environment where no predefined roles or relationships among participants exist. This is in stark contrast to our problem domain. Our domain assumes the existence of a predefined relationship between participants(i.e. in the form of roles). The paper also proposes a language (i.e. Context Condition Language) to express contextual constraints. As depicted in Figure 3.6, it introduces an associated architecture, along with its components, to support this vision of access control. The architecture supports two services (i.e. infrastructures): the Context Information Service (CIS) and the access control service. The CIS infrastructure provides functional blocks for context management tasks, such as context collection, context reasoning, etc. Whereas the access control service is responsible for access control decision-making.

The main contribution in this architecture that has not been found in the

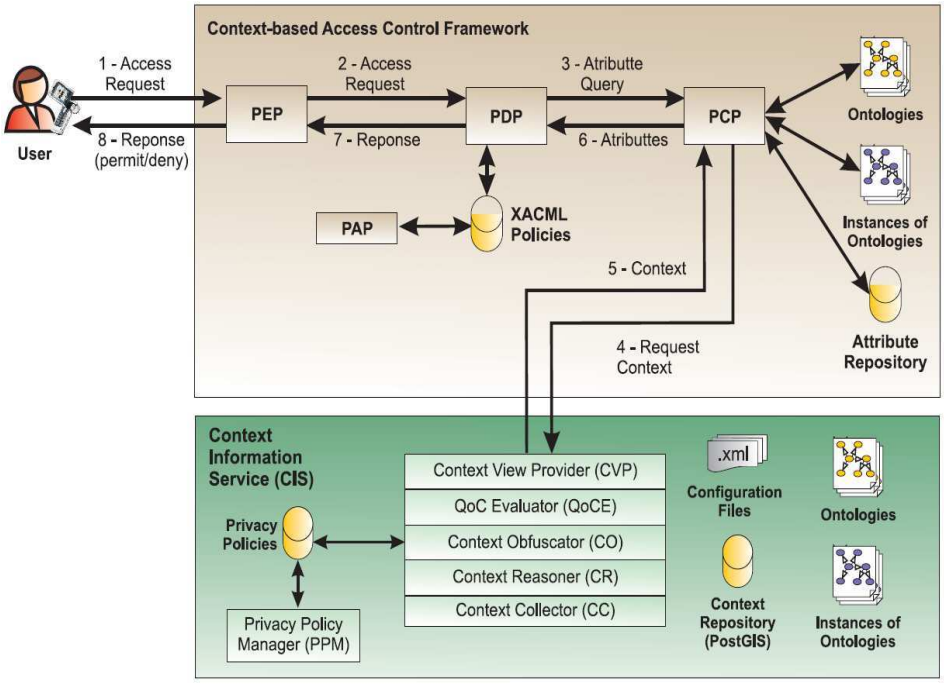


Figure 3.6: The Generalised Context-Based Access Control [9]

previous ones is: the QoC Evaluator. This component is responsible for evaluating the quality of context provided by different contextual providers. This point, as stated in Section 1.2, is crucial in context-aware access control. However, the main concern in this architecture is that it only supports one Policy Decision Point (PDP). This means that there is only a single point of decision-making. Thus, incorporating an external PDP for, for instance, a location-aware access control is not possible. In addition, while evaluating an access request, this single PDP needs to be aware of all types of contextual information used. Such a design principle may limit extensibility and flexibility.

3.3.2 Activity-Based Access Control

Another stream of access control is centred on the concept of *activity*. Many activity-based access control proposals exist in the literature (e.g. [33, 56, 57, 58, 59, 60]). In fact, the activity-based computing is a new computing paradigm that

is more suitable for UbiComp environments [56]. For example, Task-Activity Based Access Control (TABAC) focuses on access control in an environment where an activity is performed by multiple organisations [58]. In TABAC, a process is composed by activities that are associated with tasks. Permissions are dynamically assigned to tasks. A process can dynamically claim permissions as a result of its activities interactions. In other words, permissions are assigned to a task that could be linked to one or more activities. Those activities are the building block of a particular process.

Activity-Oriented Access Control (AOAC) [60] is another example of activity-based access control solutions. In AOAC, a user is allowed to carry out a certain activity if the user holds a set of required permissions. AOAC uses a PEP and a PDP. The PEP enforces access control decisions, while the PDP is a decision point at which access control decisions are made. In addition, AOAC proposes an Activity Recognition Manager that recognises users' activities. However, the paper is not clear about how activities are recognised. In fact, activity recognition is one of the major challenges in designing such a solution.

3.3.3 Using Trust to Control Access to Resources

The idea of using trust in controlling access to sensitive resource objects has been around in the literature for a while. However, most of the proposals, such as the ÆTHER model [61] and the trust-based access control model [62], are designed for an open dynamic UbiComp environment where no predefined relationships exist. For example, the ÆTHER model is designed to address trust establishment and access control management in UbiComp. It addresses the problem of trust by allowing members of the attribute authority sets, defined by service providers, to issue credentials for the corresponding attributes that can be used to access

protected resources. Attribute authority sets membership is dynamic, thus providing a distributed administration of trust amongst different authority domains. However, ÆTHER does not address how contextual information is collected and fed into the authorisation engine.

Another relevant piece of work is proposed in [63]. The work is motivated by the need for a new access control model that not only considers the contextual information but also accommodates the effect of the contextual information on the overall risk level of the underlying system. It uses some risk assessment tools to govern access control decision-making. The risk assessment process conducted in this model is online and covers confidentiality, integrity, and authentication. In other words, whenever an access request is received, the system performs an online risk assessment process to evaluate the potential risks related to confidentiality, integrity, and authentication before an access control decision is made. However, the work does not evaluate the performance of the model. As, in this model, the risk assessment process is performed online (i.e. when an access request is received) its impact on the access delay may not be negligible. In addition, it is not clear in this work how a resource object with different sensitivity levels are considered in the risk assessment process and how to fed this into access control decision-making.

3.4 What is Still Missing?

The way in which CAAC uses the contextual information in access control is troublesome. CAAC uses the contextual information as direct constraints that govern UA and PA functionalities. Many problems exist in such a way of using the contextual information. In summary, These problems are:

1. Limited set of contextual attributes: Proposals such as [37, 38, 39, 41, 42,

43, 44, 45, 47] can only cater for a small number of contextual attributes (i.e. spatio-temporal contextual attributes).

2. Limited generality, extensibility, and flexibility: A CAAC-based solution could be seen as an Access Control Infrastructure (ACI) that communicates with a Context Management Infrastructure (CMI). The ACI is responsible for authenticating users, making access control decisions and, finally, enforcing the access control decisions. Whereas the CMI provides contextual services such as context acquisition, storage, interpretation, protection and provisioning. Since CAAC *directly* uses the contextual information to control access to a protected resource object, CAAC is considered context-dependent. That is, an access control policy in CAAC is expressed in terms of the contextual information used. The corresponding authorisation engine uses the policies to grant/deny access to a protected resource object. Thus, changing the set of contextual information used will result in emphatic change in the underlying authorisation engine. This actually indicates a tight coupling between the two infrastructures (i.e. ACI and CMI) that may reduce the generality, extensibility, and flexibility of the overall access control service. For example, adding a new contextual information may require the underlying access control system to be re-engineered [64].

3. Overlooking the potential correlation among contextual attributes: CAAC overlooks the potential correlation among multiple contextual attributes and their composite effect on the authorisation decision. For example, a partial permission set may be released to a user who is accessing a certain protected resource object from a high risk location, whereas a full permission set may be released if the user is accessing the resource object from a secure location. However, the high risk location with the use of a strong

authentication token (i.e. PKI credential) may be equal to the secure location with the use of a less secure authentication token (i.e. user-name and password pair). This example indicates that the access location contextual attribute and the authentication token are correlated. Such a correlation is not explicitly considered in CAAC.

4. Overlooking the context provider's trust level. In other words, the level of trust in the provided contextual information is not used in controlling access to sensitive resource objects. This is, in fact, an important drawback in CAAC, as the context provider is assumed to be trusted (i.e. part of the Trusted Computing Base (TCB)).

3.5 The Best Way Forward

The level of impact, or potential risk, of an unauthorised access is closely related to the sensitivity level of the requested resource object. The higher the object's sensitivity level, the higher the level of impact should the object be accessed by an unauthorised entity. To provide an effective level of protection, while at the same time not introducing unnecessary overheads, the applied security protection level should be linked to the sensitivity level of the objects under protection. One way to achieve this *just-enough-security* protection is to link an authorisation decision to the Level of Assurance (LoA) in identifying the entity requesting access to an object. In other words, the higher the requested object's sensitivity level, the higher the Requester's LoA (RLoA) a user has to satisfy before granting access to the resource object.

A user's RLoA may also be influenced by the user's contextual information (i.e. both static and dynamic). In other words, we could design an access control solution that takes into consideration, not only a user's static contextual

attributes, but also the dynamic contextual attributes. The access control solution derives RLoA values, based on the static and dynamic contextual attributes, then feeds the RLoA values into the authorisation engines for a proper control of resources in UbiComp environments. In this way, we could adapt an access control decision in response to the resource object's sensitivity level and the changes in the user's LoA-affecting contextual information, thus achieving fine-grained access control.

Our proposal is summarised as follows: -

1. The resource objects are classified into groups based on their sensitivity levels. This is done as an off-line process that could be repeated if necessary. Thus, every resource object will have an Object's LoA value (OLoA) that denotes the required level of assurance a user has to satisfy before releasing the resource object. It is worth emphasising that the OLoA value is computed with respect to the object sensitivity level[2].

2. On every access request of a particular user, the contextual information of the user (i.e. both static and dynamic) is used to compute a LoA value in the identity of the user (i.e. RLoA).

3. The authorisation engine will then compare OLoA against RLoA. If $RLoA \geq OLoA$, then access is granted. Otherwise, access is denied.

4. The user's contextual information is monitored, and any change in such information will be captured and the access control decision will need to be re-assessed based on the new contextual information.

The use of LoA introduces a level of abstraction between the contextual information used and the underlying access control model. Thus, we could loosen

[2]See Subsection 4.4.2 for potential methods to achieve this

the tight-coupling between the two infrastructures (i.e. ACI and CMI). This could make the solution easily applicable in any UbiComp application domain. Moreover, the solution allows to accommodate, virtually, any set of contextual information without the need to modify the underlying access control system. We believe this is the best way to advance access control in UbiComp environments.

3.6 Chapter Summary

This chapter has surveyed some of the access control models that are proposed for UbiComp environments. In particular, it has critically analysed the CAAC-based proposals, such as GRBAC, SRBAC, TRBAC, DRBAC, and CoCoA. The most notable weakness in these proposals is the tight-coupling between ACI (i.e. responsible for access control decision-making) and CMI (i.e. responsible for contextual information management). As a result, changing the content of the contextual attribute set requires a significant modification in the ACI. In addition, some CAAC-based solutions only consider a limited set of contextual information (e.g. location and time). The potential correlation among multiple contextual attributes have also been overlooked. Recent access control proposals (e.g. the generalised context-based access control model, activity-based models, and trust-based models) have also been investigated. The gap between the issues addressed in these proposals and what is addressed in this thesis has also been identified. For example, some of these models are designed for an open UbiComp environment, which assumes no prior relationships amongst the entities involved exist. Thus, the focus is on how to establish trust under such an assumption. However, our problem scope is how to achieve just-enough security in UbiComp, in adaptation to a resource sensitivity level, while maximising flexibility and extensibility of the solution. We have outlined our vision to design an access control solution that

overcomes the existing solutions limitations.

There are some concerns regarding the proposed access control approach such as how to determine the LoA-affecting contextual information, and how to compute the RLoA value. These concerns will be covered in the following chapter.

Chapter 4

Context-Risk-Aware Access Control (CRAAC)

4.1 Chapter Introduction

As explained in Chapter 3, CRAAC achieves LoA-linked access control. In other words, based upon the risk level in the underlying access environment and/or the sensitivity level of the resource object requested, CRAAC requires an access requester to satisfy a minimum level of assurance. The level of assurance is related to the requester's contextual information. This chapter describes the CRAAC vision in detail. It identifies the contextual attributes that may affect a requester's level of assurance. It analyses the mutual relationships among the attributes and proposes methods to accommodate the relationships. Thus, at run-time, CRAAC can use the methods to dynamically derive an aggregate level of assurance (i.e. RLoA) for a given requester based upon the requester's contextual information. Then, it uses the RLoA to govern access control decision-making for the requester.

The remaining part of this chapter is structured as follows: -

- Section 4.2 introduces the CRAAC model and its vision in supporting LoA-linked access control.
- Section 4.3 identifies four contextual attributes that may have direct impact on a user's RLoA and shows how the corresponding LoA may be computed.
- Section 4.4 explores the possible LoA-to-Weight conversion methods.
- Section 4.5 identifies two mutual relationships among multiple contextual attributes and shows how to capture those relationships. In other words, it introduces two situations where the RLoA aggregation may differ: *Elevating* and *Weakest-link*.
- Section 4.6 outlines CRAAC four modes of working and the rationale behind the design of those modes.
- Finally, Section 4.7 summarises the chapter.

4.2 CRAAC Vision

The level of impact/risk of an unauthorised access depends on the sensitivity level of the requested resource object. The higher the object's sensitivity level, the higher the level of impact should the object be accessed by an unauthorised entity. To provide an effective level of security, while at the same time not to introduce unnecessary overheads, the applied security protection level should be linked to the sensitivity level of the objects under protection. One way to achieve this just-enough-security protection approach is to link an authorisation decision to the LoA in identifying the entity requesting access to an object. In other words, the higher the requested object's sensitivity level, the higher LoA a user has to satisfy when requesting the object. A user's LoA (i.e. RLoA) is derived

based on the user's contextual information. Thus, any change in the contextual information may also trigger a change in the RLoA of the user. The CRAAC model is designed to realise this vision of LoA-linked access control.

CRAAC could be seen as a MAC solution where an object is labelled with a required level of sensitivity and a subject is assigned a clearance level (i.e. RLoA). Nevertheless, CRAAC differs in the way that it does not statically assign an RLoA value to a subject (i.e. as MAC does). Rather, it computes the RLoA value dynamically for each access request.

In CRAAC, resources/services are classified into object groups each with a distinctive OLoA value. The determination of the OLoA value of an object group is based on the sensitivity level of the object group. This can be performed through an off-line risk assessment process. The proposal described in [63] is an interesting work and the same method could be used in CRAAC as well. In other words, identifying the impact of releasing a resource object on the loss of availability, confidentiality, and integrity. In general, the assessment identifies risks, evaluates their potential impacts, and maps the identified risks to an appropriate assurance level (i.e. OLoA). The OLoA of a resource object is actually the minimum LoA requirement, requested by the object, a subject has to satisfy to gain access to the resource object. The more sensitive the object is, and/or the higher the potential impact of an unauthorised access, the higher the OLoA. In detail, CRAAC access control decision-making process follows the following steps:

1. A resource provider, or a central authority, specifies an OLoA value for each resource object under his/her management. The OLoA value specification is determined based upon the object's sensitivity level and is used as a threshold to control access to the resource object.

2. When CRAAC receives an access request, it uses the surrounding contextual

information to derive a subject's RLoA value.

3. The RLoA value is then compared against the OLoA value of the required object. The access request is granted [1] iff RLoA $\geq$ OLoA.

One of the challenging tasks for realising this vision of LoA-linked access control is how to derive an RLoA value for a given access request based upon the requester's real-time contextual information. To achieve this, there is a need to: 1) identify a set of contextual attributes that may have an impact on the degree of certainty (i.e. LoA) that the access request is from an entity that it claims to be from, 2) investigate, analyse, and define the respective assurance levels for those contextual attributes, and 3) devise a method that can derive the RLoA value based upon the contextual attributes' LoA values. The following gives details on how to perform such tasks.

4.3 Contextual Attributes Identification and LoA Determination

There is a number of factors that can increase the risk of unauthorised access, e.g. weak authentication protocol/token, less trustworthy access location, poor access history, unprotected communication channels, etc. As a proof of concept, this thesis focuses on the following four contextual attributes:

- Electronic Authentication Token.
- Access Location.
- Channel Security.

[1] Provided that the access right is included in the subject's role permission set (i.e. traditional RBAC model)

- Access History.

4.3.1 Electronic Authentication Token

Many factors in an electronic authentication process may affect the assurance level (i.e. LoA) in verifying a claimed identity. These include identity proofing, credential management, record keeping, auditing, authentication protocols, and token types. The assurance levels of some of these factors are achieved through procedural and process governance, while others may be left to the requesters' decision. For example, a requester may choose to use a particular authentication credential when making an access request. As the focus here is on the derivation of an authentication LoA and on linking it to the authorisation decision making, the procedural factors (i.e. user registration, credential management, storage procedures, etc) are excluded from the LoA derivation. In other words, we only consider the effect of the types of electronic credentials/tokens, collectively called eTokens, on RLoA. Different eTokens provide varying levels of assurance in entity identification. To quantify that degree of confidence, we introduce the notion of LoA_{eToken}.

Definition 1 *LoA_{eToken} refers to the service provider's degree of confidence (i.e. assurance) that an eToken presented by a subject is linked to the subject's identity.*

The eToken types versus their assurance levels have been recommended by NIST [1], as shown in Table 4.1. NIST recognises the token types of hard tokens, soft tokens, one-time password (OTP) device tokens, and user-name/password pairs. NIST defines four levels (i.e. from 1 to 4, with Level 4 the most secure one) of LoA_{eToken} that corresponds to these tokens.

The research conducted in this thesis adopts the NIST standard for LoA_{eToken}. In other words, at run-time, a subject may choose any of the four authentication

Table 4.1: eToken Types Versus LoA_{eToken} [1]

Token Type	**Level 1**	**Level 2**	**Level 3**	**Level 4**
Hard Token	✓	✓	✓	✓
One-time Password Token	✓	✓	✓	
Soft Token	✓	✓	✓	
Password Token	✓	✓		

methods (i.e. eTokens). The LoA associated with the eToken can be determined by Table 4.1. Changing an authentication method token type during the course of an access session may also change the corresponding LoA.

4.3.2 Access Location

Authentication services are of two main types; one is e-authentication by which a user is identified through the use of an eToken, and the other is physical authentication (p-authentication) by which a user is identified through the use of biometrics, sensors, or location based services. CRAAC recognises both of these authentication service types. This is because, firstly, a combined use of e-authentication and location-based p-authentication may not only provide optional services to users but also a more reliable user identification. Secondly, the proliferation of the location-aware services in UbiComp requires the access control service to accommodate the location information as well. Therefore, in addition to the eToken attribute, we introduce another authentication attribute called Access Location (ALoc). The assurance level of ALoc, LoA_{ALoc}, is defined below.

Definition 2 *LoA_{ALoc} refers to the degree of confidence/assurance in a subject's claimed access location.*

Depending on the application context, there are various approaches to represent the location alternatives [65, 66]. As our focus is on the degree of confidence

in a claimed location and as a proof of concept, we use the *zone* representation method [65, 39] to describe different location alternatives. Table 4.2 shows some possible location alternatives versus the corresponding assurance levels. The table is meant to illustrate how the location LoA values may be determined. Unlike the case of eToken, there is no international consensus on defining LoA_{ALoc} values.

Table 4.2: A Sample Location Information Versus LoA_{ALoc} [2]

Location Alternative	**Level of Assurance**
$Zone_0$	$Level_0$: public area which does not have any provision for p-authentication.
$Zone_1$	$Level_1$: semi-public area which uses p-authentication to identify a group of users, e.g. through the use of a shared building key.
$Zone_2$	$Level_2$: personal area – access to this zone is controlled by the use of a locker key owned by a single user or a sensor based user identification (e.g. RFID).
$Zone_3$	$Level_3$: secured personal area – this zone uses some strong form of physical identification method that is less vulnerable to theft or loss than locker keys, e.g. Biometrics (physical) authentication facility.
$Zone_4$	$Level_4$: highly secured personal area – this zone may use multiple physical authentication methods.

Both authentication attributes (i.e. eToken and ALoc) make a direct contribution to the overall assurance level in identifying a user. They could be seen as a two-factor authentication mechanism. For example, a service provider may use two authentication services to identify a user; one is the user-name and password method, and the other is a biometric-based location authentication. As a password token is more vulnerable to guessing attacks than a PKI credential, using it inside a secure location with a biometric physical authentication facility may be comparable, in terms of authentication assurance level, to a PKI credential used in a public area.

To quantify the overall assurance level of the two additive attributes, we introduce the notion of LoA_{authN}.

Definition 3 *LoA_{authN} is the overall confidence/assurance level associated with the composite authentication solution consisting of token-based e-authentication (i.e. eToken) and location-based p-authentication (i.e. ALoc).*

The introduction of the authN attribute, and its corresponding LoA definition, is a major focus of this thesis. The authN attribute may be extended to abstract all contextual attributes that may impact the level of assurance in identifying a subject. In this way, the number and the types of the used authentication methods are not significant on CRAAC access control decision-making. Rather, authN is the considerable factor that affects access control decision-making. Currently, the authN attribute only encompasses the eToken and ALoc contextual attributes. As technology advances, further research may be needed to identify other potential contextual attributes that may have an impact on the level of assurance in the authentication method used by a subject.

4.3.3 Communication Channel Security

The level of security protection of the communication channel, linking a subject and a service provider, may also influence the risk level of unauthorised access. For instance, if a channel is vulnerable to eavesdropping attacks, some credentials sent over the channel may experience a high risk of being compromised. In addition, the requested data may experience a high risk of being disclosed to unauthorised entities via channel interceptions. For these reasons, we introduce the Channel Security (CS) attribute.

Definition 4 *LoA_{CS} refers to the degree of confidence/assurance in the channel,*

linking a subject and a service provider, in protecting the confidentiality of data transmitted over it.

Similar to the ALoc attribute, the CS attribute does not have an international consensus on its assurance level definition. Table 4.3 describes an exemplar setting of a 5-level LoA_{CS} mimicking the NIST's eToken LoA definition.

Table 4.3: An Examplar Setting of LoA_{CS} Values

LoA_{CS}	**Description**
$Level_0$	This attribute is disabled, or not used.
$Level_1$	Little or no confidence in channel security.
$Level_2$	Some confidence in channel security.
$Level_3$	High confidence in channel security.
$Level_4$	Very high confidence in channel security.

4.3.4 Access History

A subject's access history (AH) is an important indicator of a subject's trustworthiness. A user with repeated authentication failures and/or repeated authorisation rejections should score a low access history assurance level. By accommodating AH in access control decision-making, CRAAC may help to deter malicious attempts and encourage good behaviour among subjects. A user's access history attribute stores information about the user's history related to his/her authentication and authorisation outcomes for a past period. The corresponding LoA value could be computed, mimicking the NIST work for eToken, as depicted in Table 4.4. The LoA of a subject's AH is defined as follows:

Definition 5 *LoA_{AH} refers to the degree of confidence/assurance in the access history of a subject. This encompasses the degree of assurance in authentication and authorisation history.*

Table 4.4: An Examplar Setting of LoA_{AH} Values

LoA_{AH}	**Description**
$Level_0$	This attribute is disabled, or not used.
$Level_1$	Little or no confidence in the subject's access history.
$Level_2$	Some confidence in the subject's access history.
$Level_3$	High confidence in the subject's access history.

4.4 LoA-to-Weight Conversion (L2WC) Method Selection

4.4.1 Rationale and Selection Criteria

The attributes' LoA (i.e. LoA_{eToken},LoA_{ALoc}, LoA_{CS} and LoA_{AH}) discussed earlier are ranks (e.g. $level_1$). A LoA rank needs to be converted into rating (i.e. weight) that corresponds to its significance before it can be used to derive a subject's RLoA. In CRAAC, this process is called LoA-to-Weight Conversion (L2WC).

There are multiple methods that could be used to perform L2WC. However, the choice of an appropriate conversion method should be in-line with the general requirements of UbiComp environments. This UbiComp conformance focuses on non-intrusiveness and performance factors. Since the conversion is performed on-line, based on the changes of the surrounding contextual information, the conversion method should be efficient in terms of the time it takes to accomplish the conversion. In addition, the use of the capability-restrictive devices in UbiComp prefers the use of a computationally lightweight conversion method. In certain cases, the device that performs the conversion may be a PDA with a low computational power and/or a short battery life. A conversion method that consumes a lot of resources is not appropriate in this environment. Moreover, the process should be systematic enough to eliminate human intervention in its

calculations. Non-intrusiveness is a fundamental requirement in UbiComp [13]. This may be achieved by utilising an "objective" conversion method that does not depend heavily on the personal views of, for instance, a central authority and his/her understanding of the problem.

4.4.2 L2WC Methods

There are multiple methods to perform the L2WC. This includes the Analytical Hierarchy Process (AHP), Rank-order, and fuzzy methods. As the fuzzy method lacks an acceptable ranking method [67], our discussions below focus on AHP and Rank-order methods.

4.4.2.1 Analytical Hierarchy Process

AHP, proposed by Thomas Saaty [3], is based on mathematics and psychology. It compares a list of alternatives based on multiple selection criteria. In other words, AHP can be used to make complex decisions that involve multiple criteria. It provides assistance to a decision-maker in identifying and weighting the selection criteria. Ranking and weighting alternatives in order from most to least significant is an important application of AHP, which is our main aim in studying AHP. The way in which AHP weighs alternatives can also be used to weigh the LoA ranks in our problem context. As the main aim here is to show how AHP can be used to perform L2WC, the following only illustrates the operations required for performing such a task.

Consider a decision manager needing to make a decision about which software product to buy for an organisation. In surveying the market, there are 5 competing companies with 5 different products: A, B, C, D, and E. To select the best product, the manager selects the AHP technique. AHP requires the decision

manager to form a hierarchy that models the problem. The hierarchy should contain the alternatives (i.e. A, B, C, D, and E products) at the base level, and it should encompass all selection criteria required to evaluate these alternatives against. Figure 4.1 shows an example hierarchy, where the decision manager lists a set of four criteria that help to discriminate between those competing products. AHP in general allows a decision manager to choose a selection criterion that may be tangible, intangible, accurately measured, or approximately measured. As shown in Figure 4.1, *start-up time*, *search speed*, *cost*, and *ease of use* are the selection criteria the decision manager has identified for the software product selection.

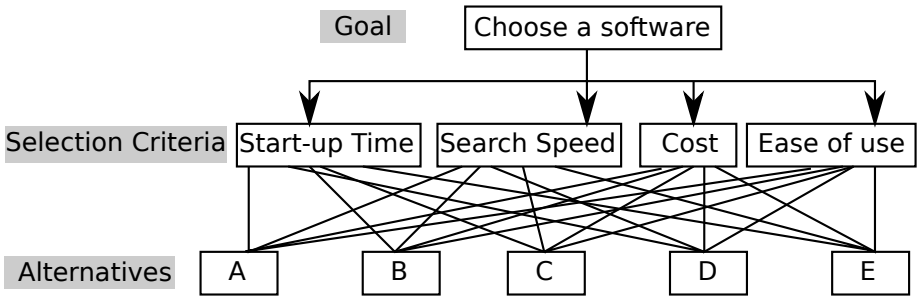


Figure 4.1: The AHP Hierarchy for the Software Selection Problem

After constructing the AHP hierarchy, the importance of the various selection criteria has to be evaluated. This is done apart from the software product alternatives. For example, given the selection criteria level in Figure 4.1, is the "start-up time" more important than the "search speed" in selecting a software product? This process actually establishes priorities among the selection criteria (i.e. ranking) with respect to achieving the goal (i.e. selecting the best software product). To perform the ranking, all the selection criteria of the hierarchy will be compared to one another in a pair-wise comparison (i.e. two selection criteria at a time). For a systematic calculation of the selection criteria relative weights, Saaty has defined 9 levels of importance as described in Table 4.5. There are

Table 4.5: The AHP Importance Rating Scale [3]

Importance intensity	Definition	Explanation
1	Equal importance	Two factors contribute equally to the objective
3	Somewhat more important	Experience and judgement slightly favour one over the other
5	Much more important	Experience and judgement strongly favour one over the other
7	Very much more important	Experience and judgement very strongly favour one over the other
9	Absolutely more important	The evidence favouring one over the other is of the highest possible validity
2,4,6,8	Intermediate values	When compromise is needed

multiple expectations from the relative weight calculation process, such as 1) the importance of all the selection criteria must add up to 1, which is the priority of the main goal, and 2) the relative importance of the selection criteria (C_i, C_j) must equal to 1, where $i = j$. The reason for this is that each selection criterion is as important as itself. In conducting the pair-wise comparison, the decision maker can use concrete measures (i.e. objective) or personal judgement (i.e. subjective). For instance, the "cost" and the "start-up time" could be measured with absolute certainty. The "start-up time", for instance, can be quantitatively measured by computing the time it takes to start up the application. On the other hand, the "ease of use" may be judged subjectively based on the decision manager's own perception of the software application. The results of the pair-wise comparisons are placed in a matrix, as depicted in Table 4.6. The matrix shows, for instance, that the "search speed" is slightly more important than the "start-up time". This is encoded as 3 and 1/3 in the highlighted cells. The numbers in the matrix represent the relative importance of the selection criteria;

Table 4.6: The AHP Relative Importance Matrix for Software Selection

	start-up time	search speed	cost	ease of use
start-up time	1	1/3	5	1
search speed	3	1	5	1
cost	1/5	1/5	1	1/5
ease of use	1	1	5	1

In other words, they represent the ranks of the selection criteria. The next step in AHP is to calculate the relative weights for each selection criterion given the ranks determined in the matrix. This, as shown in Table 4.7, is performed by taking each entry in the original matrix, shown in Table 4.6, and dividing it by the sum of the column it appears in. For example, the highlighted ("search speed“,

Table 4.7: The AHP Relative Weights for Software Selection

	start-up time	search speed	cost	ease of use	Average
start-up time	0.1923	0.1316	0.3125	0.3125	0.2372
search speed	0.5769	0.3947	0.3125	0.3125	0.3992
cost	0.0385	0.0790	0.0625	0.0625	0.0606
ease of use	0.1923	0.3947	0.3125	0.3125	0.3030

"cost“) entry is computed as $\frac{5}{5+5+1+5} = 0.3125$. A new matrix is constructed, as shown in Table 4.7, that represents the relative weights of the selection criteria. The matrix, for example, shows that the "search speed“ is the most important selection criterion, contributing about 40% to the overall goal. As far as this thesis is concerned, up to this point AHP can be used to perform the L2WC. The rest of the AHP operations (i.e. selecting an alternative) are beyond the thesis scope.

4.4.2.2 Rank-Ordered Weights-Based Methods

The rank-order method is used to generate weights for different alternatives. Rank-order centroid (ROC) [68], Rank Reciprocal (RR), Rank Exponent (RE), and Rank Sum (RS) [69] are examples of the rank-order method. This method

is often used to solve Multiple Criteria Decision Analysis (MCDA) problems. It takes a set of attributes ordered by importance (i.e. ranks) and converts them into a set of approximated weights (i.e. ratings). It is worth emphasising that sometimes it may not be realistic to determine the precise weights of the attributes [70].

The main difference amongst RS, RR, RE, and ROC is the formula they are using to approximate the weights of the ranks. RS uses the following formula [69]:

$$w_i = \frac{N - R_i + 1}{\sum_{j=1}^{N}(N - R_j + 1)} \tag{4.1}$$

Where N is the number of attributes, R_i is the rank position of the attribute. RR uses another formula: [69]:

$$w_i = \frac{1/R_i}{\sum_{j=1}^{N}(1/R_j)} \tag{4.2}$$

RE uses [69]:

$$w_i = \frac{(N - R_i + 1)^z}{\sum_{j=1}^{N}(N - R_j + i)^z} \tag{4.3}$$

where z is the weight of the most important attribute on a 0-1 scale. This is an additional piece of information required in RE to compute the weights of the corresponding attributes. It is worth noting that if z is 1, the RE defaults back to RS, and if z is 0 that corresponds to the equal weights case.

ROC, originally proposed by Barron in [68], has an appealing theoretical foundation for its derived weights [71]. It derives weights through a systematic analysis of implicit information in the ranks, which would give an accurate outcome [70]. Using the ROC method, the weights are derived from a simplex

$w_1 \geq w_2 \geq \ldots \geq w_n \geq 0$ restricted to:

$$\sum_{i=1}^{n} w_i = 1 \tag{4.4}$$

where n is the number of attributes (system cardinality). The vertices of the simplex are $e_1 = (1, 0, \ldots, 0), e_2 = (\frac{1}{2}, \frac{1}{2}, 0, \ldots, 0), e_3 = (\frac{1}{3}, \frac{1}{3}, \frac{1}{3}, 0, \ldots, 0), \ldots, e_n = (\frac{1}{n}, \frac{1}{n}, \ldots, \frac{1}{n})$. The coordinates of the centroids (i.e. weights) are calculated by averaging the corresponding coordinates of the defined vertices [70]. In general, the weight of the k^{th} most important attribute out of n attributes is calculated as:

$$w_k = \frac{(\sum_{i=k}^{n} 1/i)}{n} \tag{4.5}$$

For example, consider a set of four attributes: A, B, C, and D ranked as first (i.e. most important), second, third, and fourth (i.e. least important), respectively. ROC would compute the weights as: $A = (1 + 1/2 + 1/3 + 1/4)/4 = 0.5208$, $B = (0 + 1/2 + 1/3 + 1/4)/4 = 0.2708$, $C = (0 + 0 + 1/3 + 1/4)/4 = 0.1458$ and, $D = (0 + 0 + 1/4)/4 = 0.0625$.

4.4.3 Choosing L2WC Method

Table 4.8 summarises the potential conversion methods surveyed in this research. The RE method is not appropriate selection, since it requires extra information (i.e. the weight of the most important attribute) to compute the weights. In our L2WC problem, such information can not be provided. As seen in the table, the rank-ordered weights-based methods (i.e. RR, RS, and ROC) show more accurate results than the AHP method. AHP sometimes produces unreasonable weights due to its subjective way of weights determination. For example, if A is more important than B, which is more important than C then A is expected to be

more important than C (i.e. transitivity). In AHP, as the pair-wise comparison is performed on two items at a time, a decision manager may explicitly state that C is more important than A, which is not sensible. Pair-wise comparison in AHP is a kind of a stateless process that can not link a current pair-wise comparison with a previous one. In other words, AHP uses pair-wise comparison that may not acknowledge the transitivity of relative importance among factors (i.e. selection criteria). No tool in AHP could be used to prevent such a possibility.

Table 4.8: L2WC Methods Comparison

	Accuracy	**Complexity**	**Ease of Implementation**	**Weights Calculation**	**Foundation**	**Extra Knowledge**
AHP	Concern	$O(n^2)$	✓	(Objective,non-intrusive), (Subjective,intrusive)	Math & psychology	✗
RR	Accurate	$O(n)$	✓	Objective, non-intrusive	Ad Hoc [70]	✗
RS	Accurate	$O(n)$	✓	Objective, non-intrusive	Ad Hoc [70]	✗
RE	Accurate	$O(n)$	✓	Objective, non-intrusive	Ad Hoc [70]	✓
ROC	Most accurate	$O(n)$	✓	Objective, non-intrusive	Systematic Analysis of ranks	✗

In addition, AHP can create a rank reversal phenomenon, where adding irrelevant alternatives may cause a reversal in the ranking [72]. Moreover, the static 9-levels rating system proposed by Saaty does not always cope with some marginal differences of importance [73]. The time complexity of the rank-ordered

weights-based methods is linear (i.e. $O(n)$), where n is number of the contextual attributes used. In contrast, in AHP, the number of pair-wise comparisons is $\frac{n\times(n-1)}{2}$, where n is the number of attributes. This means the complexity of AHP is of $O(n^2)$. In other words, the number of comparisons conducted in, for instance, ROC is less than that of AHP [67]. Based on the above considerations, AHP is excluded as a choice for L2WC.

Comparing ROC against other rank-ordered methods, ROC is more accurate, and provides an efficient and appropriate implementation tool [70]. The RR and RS weights calculation process is Ad Hoc [70], whilst in ROC it is based on a systematic analysis of ranks. Based on the above considerations, ROC is chosen as the L2WC method in this research. To further reduce the time spent on calculating weights, a static conversion table can be used. In other words, the weights of the contextual attributes can be calculated off-line and a table is preloaded with the weights. On receiving an access request, the access control service can look up the static table for the weights of the corresponding contextual attributes instead of calculating the weights in real-time.

4.5 Requester's LoA Aggregation at Run-Time

As mentioned earlier, the assurance level in identifying a subject may be influenced by multiple attributes, either directly (e.g. eTokens, ALoc, and AH) or indirectly (e.g. CS). To quantify the assurance level as influenced by the combination of a subject's multiple contextual attributes, the RLoA notion is introduced. We define RLoA as follows:

Definition 6 *RLoA refers to an overall LoA in identifying a subject based upon the subject's contextual information that is associated with the subject's multiple contextual attributes (i.e. eToken, ALoc, CS, and AH).*

The derivation of RLoA depends on the types of the contextual attributes and the correlation, or the mutual relationships, amongst the contextual attributes. Formally, given a set of contextual attributes $(A_1, A_2, \ldots, A_n)$ and their associated assurance levels $(LoA_{A_1}, LoA_{A_2}, \ldots, LoA_{A_n})$, RLoA can be expressed using a generic function, f, as:

$$RLoA = f(LoA_{A_1}, LoA_{A_2}, \ldots, LoA_{A_n}) \tag{4.6}$$

f is determined by the relationship among the multiple contextual attributes. We have identified two types of relationships: *Elevating* and *Weakest-link*.

4.5.1 Elevating Relationship

In the elevating relationship, the combined use of two or more contextual attributes may result in the overall confidence level being higher than that provided by any of the individual contextual attributes. The concept of elevating security is used by Microsoft in Windows Server 2003 to enable regular users to install applications even if they do not have the required permissions [74]. In our problem context, the eToken and ALoc attributes (i.e. when used collectively to identify a user) are in an elevating relationship. In fact, a combined use of e-Token and ALoc is a two-factor authentication solution that is more reliable than using only eToken or ALoc alone. Thus, it provides a higher RLoA value.

Given that a requester has n contextual attributes, $(A_1, A_2, \ldots, A_n)$, all the attributes are in an elevating relationship, and each of the attribute has a LoA associated with it, $(LoA_{A_1}, LoA_{A_2}, \ldots, LoA_{A_n})$, where $1 > LoA_{A_i} > 0$, $i \in \{1, n\}$, then the overall assurance value (i.e. RLoA) can be calculated (using probability

theory) as [75]:

$$RLoA = 1 - (1 - LoA_{A_1})(1 - LoA_{A_2}) \dots (1 - LoA_{A_n}) \tag{4.7}$$

An advantage of Equation 4.7 is that an attribute with a higher assurance value would have a higher impact on RLoA, and an attribute with a lower assurance value would have a lower impact on the overall assurance value. Based on our knowledge and literature [76], we have observed that, among the set of attributes {eToken, ALoc, CS, AH}, only eToken and ALoc attributes are in an elevating relationship. Other attributes are in a weakest-link relationship. Thus, by applying Equation 4.7 to the eToken and ALoc attributes, LoA_{authN} can be calculated as:

$$LoA_{authN} = 1 - (1 - LoA_{eToken})(1 - LoA_{ALoc}) \tag{4.8}$$

Here, LoA_{authN} is the aggregated LoA produced by the eToken and ALoc attributes. Replacing the eToken and ALoc attributes with LoA_{authN}, equation 4.6 can be revised as:

$$RLoA = f(LoA_{authN}, LoA_{CS}, LoA_{AH}) \tag{4.9}$$

4.5.2 Weakest-Link Relationship

In the weakest-link relationship the value of RLoA is equal to the lowest attribute LoA value in the attribute value set. This is in line with the weakest-link principle in system security. This is because, even if the underlying authentication procedure is strong (thus difficult to impersonate), and the channel security has a high assurance level (thus difficult to intercept confidential information), provided that

the service provider's system is easy to break into, there will be still a high risk of compromising the server end of the identification and authentication procedure, e.g. by directly attacking the system credential stores, or by tampering with the authentication algorithm, etc. This implies that, for a set of attributes that are in a weakest link relationship, the overall assurance level should not be higher than the lowest attribute LoA involved.

For example, the attributes authN (i.e. as computed from eToken and ALoc), CS, and AH, resembles more the weakest-link relationship. The RLoA is calculated as:

$$RLoA = min(LoA_{authN}, LoA_{CS}, LoA_{AH}) \tag{4.10}$$

Where *min* is the minimum function that returns the smallest LoA value of those enclosed in the brackets. Note that the calculation of LoA_{authN} remains the same, as the eToken and ALoc attributes are in an elevating relationship due to its two-factor authentication nature.

4.6 CRAAC Modes of Working

CRAAC supports four modes of working; each suits a different application context and access control requirements. The four modes are:

- *RLoA-only Mode*: The RLoA value of multiple attributes is used to govern access control decision-making.
- *AttributeLoA-only Mode*: The individual LoA values of one or more attributes are used to govern access control decision-making.
- *Combined Mode*: Both RLoA-only and AttributeLoA-only modes are used to govern access control decision-making.

- *Basic-RBAC Mode*: This corresponds to the traditional RBAC model.

The four modes differ by which the LoA values are used to control access to sensitive resource objects. However, the basic RBAC mode is fundamental in all other modes.

4.6.1 RLoA-only Mode

In this mode, resources are classified into groups based on their sensitivity levels. The resource classification is performed transparently from the set of utilised contextual attributes. In other words, a resource provider may not need to be aware of how users will be identified and what contextual attributes are used. The resource provider only needs to specify a single minimum level of assurance to release a given resource object (i.e. OLoA). When an access request is received, the set of contextual information associated with the access requester is assessed. A corresponding RLoA value is derived based on the set of contextual information. The access will be granted iff RLoA $\geq$ OLoA. In this way, we could adapt an access control decision in response to the resource object's sensitivity levels and the changes in the user's LoA-affecting contextual information thus, achieving fine-grained access control. In addition to governing the role-permission assignment, RLoA can be used to govern the user-role assignment. In other words, in the RLoA-only mode, RLoA can be used to govern both UA and PA functionalities.

4.6.2 AttributeLoA-only Mode

In this mode of working, access to sensitive resource objects is governed by individual contextual attributes' LoA values. That is, each contextual attribute has its own AttributeLoA value, which will be used, possibly along with other attributeLoA values, to control access to sensitive resource objects. In other words,

a service provider can specify a set of LoA requirement on a particular set of contextual attributes. An access requester has to satisfy all the LoA requirements to gain access to the service.

There is a similarity between this mode of working and the CAAC-based models discussed in Chapter 3. Both use the contextual information as additional constraints to govern both UA and/or PA functions. But, instead of using contextual attribute values in the access control policy specification as in CAAC, the AttributeLoA-only mode uses the attributes' LoA values. In this way, the AttributeLoA-only mode hides the way in which contextual information is represented from the authorisation engine, which provides more flexibility. In other words, CRAAC generalises the CAAC-based model in the way that it is not required for the authorisation engine to be aware of the representation method used to express the contextual information. For example, an access control policy for a location-aware access control service that uses the "zones" representation could be the same as the one that uses an absolute positioning representation(i.e. longitude and latitude). In the AttributeLoA-only mode, CRAAC can still utilise a CAAC-based authorisation engine, used by many existing context-aware systems, as an external authorisation engine.

4.6.3 Combined Mode

This mode combines the use of the RLoA-only and AttributeLoA-only modes. For example, in certain application scenarios, an access requester may be assigned a role based on the RLoA value (i.e. UA function), whereas permissions are granted based on the individual contextual attributes LoA values (i.e. PA function), or any other combinations. CRAAC supports such an access control requirement by proposing the Combined mode of working.

4.6.4 Basic-RBAC Mode

This mode controls access to sensitive resource objects based solely on the RBAC model. Identifying an access requester is performed based on the requester's static attributes such as ID. This mode is fundamental for all other modes of working. However, it may be used only when an access control system disables the use of contextual information. For example, when a location sensor in a location-aware access control service is switched off. The access control service should, hence, use a traditional method for access control (i.e. RBAC).

4.7 Chapter Summary

This chapter has described several important design issues associated with the design of CRAAC. These issues include the identification of authentication LoA relevant contextual attributes, their LoA quantification, their mutual relationship analysis, and the derivation of RLoA. CRAAC uses RLoA as a generic attribute to capture the composite effect of a subject's contextual information on the subject's assurance level, and uses it to govern the set of permissions assigned to the subject. By linking RLoA to a resource object sensitivity level, CRAAC not only achieves context-aware but also risk-aware access control. Most importantly, through the use of RLoA, CRAAC has successfully decoupled its access control function from its contextual information management function. Thus, any change in either functional modules will only require minimum alteration in the other. This provides the flexibility and generality that CRAAC is seeking. Furthermore, CRAAC supports four modes of working to satisfy a divergent set of access control and policy specification requirements, making it applicable in a wide range of application contexts.

Chapter 5

CRAAC Design Preliminaries

5.1 Chapter Introduction

To realise the CRAAC vision discussed in Chapter 4, a CRAAC architecture is needed. This architecture should, by proposing functional blocks, support context management, access control decision-making, and LoA derivation services. This chapter describes the design principles and methods for the CRAAC architecture. Flexibility, extensibility, generality, high level functional encapsulation, and LoA-linked fine-grained access control are amongst the design principles of the CRAAC architecture. To support a LoA-linked fine-grained access control, CRAAC supports three policy types and two policy retrieval modes. Introducing the architecture will undoubtedly, increase the overall model complexity. This may affect the applicability of the CRAAC model and may hinder the acceptability of the model by an enterprise. For this reason, CRAAC should be evaluated against a well-known access control quality metric. This chapter introduces a quality metric by which CRAAC is assessed. The main elements of the quality metric are: safety and performance. To measure the CRAAC performance, a CRAAC prototype has been built. The prototype development environment will

be described in this chapter.

The remaining part of this chapter is structured as follows: -

- Section 5.2 describes the motivations and the design requirements of the CRAAC architecture.
- Section 5.3 discusses three policy types used by CRAAC.
- It also outlines two different modes of retrieving policy data from an access control policy store (i.e. *push* and *pull*).
- Section 5.4 discuses the evaluation metric by which the CRAAC model is evaluated. Moreover, it describes the CRAAC evaluation testbed.
- Finally, Section 5.5 summarises the chapter.

5.2 CRAAC Architecture: Motivation and Design Requirements

To realise the CRAAC vision of LoA-linked access control, there is a need for a supporting architecture. The architecture should acquire contextual information from different context sources, quantify the corresponding LoA (i.e. ranks), convert the LoA ranks into LoA weights (i.e. L2WC) and aggregate the LoA values into one RLoA value[1]. Then, it should feed the LoA/RLoA values into the authorisation decision engine and, finally, produce an access control decision.

The following requirements have been used in the design of the CRAAC architecture:

1. LoA-aware

   CRAAC should provide built-in services to support the use of individual

[1]Depending on the configured mode of working

LoA values (i.e. AttributeLoA values) as well as aggregate LoA values (i.e. RLoA values) in access control decision-making.

2. Extensibility and Flexibility

   The CRAAC architecture should be extensible in order to allow easy addition of new and removal of obsolete contextual attributes. Any alterations imposed on the architecture caused by such contextual attribute changes should only be refrained within the context management part of the CRAAC architecture. Generally, any change occurred in an architectural component should not significantly affect other components in the architecture. In addition, an addition/deletion of an architectural component should impose a minimum change on the rest of the architecture. This property can be achieved through functional encapsulation. This is essential for the CRAAC model to cope with the dynamic nature of the UbiComp environments. In addition, the functional encapsulation can make the CRAAC architecture flexible enough to accommodate different numbers and types of contextual attributes. The architecture should be applicable to different application contexts. It should be able to employ both internal as well as external authorisation decision engines.

3. Efficient performance

   The CRAAC architecture should perform efficiently in terms of access delays/latency that a subject has to endure before an access control decision is made. This also includes streamlining the inter-component communication to reduce the number of interactions and communication overheads. Another factor that may affect the performance of the system is scalability. That is why the performance of CRAAC is investigated when the number of

enabled roles for a particular user increases[2], the number of the contextual attributes increases[3], the access request rates increases[4], etc.

4. Standard architectural component design

   The CRAAC architecture should comply with relevant standards (e.g. XACML) in order to provide interoperability with the current solutions.

5.3 CRAAC Policy Types and Access Modes

There are two further design issues: 1) the types of access control policies supported by CRAAC, and 2) the access mode that is used to retrieve a policy from its store.

CRAAC recognises three types of access control policies: UA, PA, and resources' OLoA policies. The UA and PA policies store user-role and role-permission mappings, respectively. These are the fundamental functions of the basic RBAC model. A UA policy is relatively smaller than that of PA and it could be stored in a relational database table for efficient retrieval. A PA policy needs to be expressed in a standard access control policy language, due to its relatively big size. In this research, XACML 2.0 [77] is used to express this policy type. However, the architecture is flexible in order to allow any other policy languages to be used.

CRAAC expresses the UA policy in a 3-tuple format: $< Subject, RLoA, Role >$, or in an n-tuple format: $< Subject, LoA_1 ... LoA_n, Role >$, where n is the number of contextual attributes recognised by the system. The use of 3 or n-tuple policy format depends on the CRAAC configuration (i.e. mode of working). For example, the 3-tuple expression is used in the RLoA-only mode, whereas the n-tuple is used in the AttributeLoA-only mode.

[2]See Subsection 6.5.3 for more detail
[3]See Subsection 7.5.2 for more detail
[4]See Subsection 6.5.6 for more detail

The resources' OLoA policy specifies the minimum LoA requirement upon which a resource object can be released. This policy is stored in an XML file and is maintained independently from the other two polices. Figure 5.1 depicts a snippet of the resources' OLoA policy for a "Printer" object. The figure shows the XML file divided into two main elements: OLoA and Individual-OLoA-Set. The OLoA element contains the OLoA specification that the printer requires to be released, regardless of the type, number, or representation of the contextual attributes used. For example, for a subject to cancel the current printing task, the subject has to satisfy a LoA requirement of at least 0.48. In other words, a subject has to satisfy this constraint (i.e. RLoA $\geq$ 0.48) to gain access to the Printer no matter what contextual attributes the subject has subscribed to. This element (i.e. OLoA) is used in both RLoA-only and Combined modes.

```
</ResourceObjects>
<Resource ID="CRAAC:10"  Name="Printer">
<OLoA>
<LoA-Entry Context-Attribute ="NONE" Permission="SwitchOn"> <Value>0.04</Value></LoA-Entry>
<LoA-Entry Context-Attribute ="NONE" Permission="SwitchOff"><Value>0.04</Value></LoA-Entry>
<LoA-Entry Context-Attribute ="NONE" Permission="print"><Value>0.70</Value></LoA-Entry>
<LoA-Entry Context-Attribute ="NONE" Permission="CancelCurrentTask"><Value>0.48</Value></LoA-Entry>
</OLoA>
<Individual-OLoA-Set>
<LoA-Entry Context-Attribute ="CS" Permission="SwitchOn"><Value>0.02</Value></LoA-Entry>
<LoA-Entry Context-Attribute ="AH" Permission="SwitchOn"><Value>0.03</Value></LoA-Entry>
<LoA-Entry Context-Attribute ="ALoc" Permission="SwitchOn"><Value>0.01</Value></LoA-Entry>
<LoA-Entry Context-Attribute ="eToken" Permission="SwitchOn"><Value>0.03</Value></LoA-Entry>
<LoA-Entry Context-Attribute ="eToken" Permission="SwitchOff"><Value>0.03</Value></LoA-Entry>
<LoA-Entry Context-Attribute ="CS" Permission="SwitchOff"><Value>0.01</Value></LoA-Entry>
<LoA-Entry Context-Attribute ="CS" Permission="FaxIt"><Value>0.24</Value></LoA-Entry>
<LoA-Entry Context-Attribute ="AH" Permission="FaxIt"><Value>0.20</Value></LoA-Entry>
<LoA-Entry Context-Attribute ="eToken" Permission="FaxIt"><Value>0.40</Value></LoA-Entry>
<LoA-Entry Context-Attribute ="CS" Permission="CancelCurrentTask"><Value>0.24</Value></LoA-Entry>
<LoA-Entry Context-Attribute ="AH" Permission="CancelCurrentTask"><Value>0.40</Value></LoA-Entry>
<LoA-Entry Context-Attribute ="eToken" Permission="CancelCurrentTask"><Value>0.50</Value></LoA-Entry>
</Individual-OLoA-Set>
</Resource>
</ResourceObjects>
```

Figure 5.1: Snippet of the Resources' OLoA Policy

On the other hand, the Individual-OLoA-Set element expresses the Printer's OLoA requirements in terms of the individual contextual attribute LoA values. This element of the policy is context-aware, since it needs to know the type and the number of the contextual attributes used. This element supports the use of both AttributeLoA-only and Combined modes, since, for example, the AttributeLoA-only mode utilises individual contextual attributes' LoA values to

govern access control decision-making. Table 5.1 summarises which mode of working uses what type of access control policy.

Table 5.1: CRAAC Modes Vs Policy Files Usage

	UA	PA	Resources' OLoA: OLoA Element	Resources' OLoA: Individual-OLoA-Set
RLoA-only Mode	✓	✓	✓	✗
AttributeLoA-only Mode	✓	✓	✗	✓
Combined Mode	✓	✓	✓	✓
Basic-RBAC Mode	✓	✓	✗	✗

Two policy retrieval modes are recognised by CRAAC: *pull* and *push* modes [78]. In the *pull* mode, an access control policy is retrieved from the corresponding store on demand. In other words, on receiving an access request, CRAAC opens the corresponding access control policy file, parses it, and verifies the request against the policy rules. In the *push* mode, on the other hand, the access control policy is pushed into the system before receiving any access request. In other words, in this access mode, CRAAC is pre-loaded with the policies when it is initialised. The use of both modes in CRAAC and their implications on the CRAAC performance will be reported in Chapters 6 and 7.

5.4 CRAAC Evaluation

Evaluating the CRAAC model is required; as an access control system grows in size, the system complexity grows as well and more overheads may be imposed on the system. In conducting the CRAAC evaluation, the results of the evaluation will be compared to those from the traditional RBAC system (i.e. as a reference model). In fact, evaluating an access control system is challenging. To our best knowledge, the NIST quality metric [4] is, perhaps, the most well-known metric

that is used for such a purpose.

The NIST quality metric, as depicted in Table 5.2, does not suggest any tangible measurements, nor does it propose discrete benchmarks, for each metric element. The NIST quality metric contains elements that verify the administrative capabilities, administrative cost, policy coverage, extensibility, and performance quality. The selection of a metric element depends on a decision manager and the access control system in hand. In this thesis, the NIST quality metric is adopted as a common basis to assess the CRAAC model. The main focuses of the CRAAC assessment is on performance and safety elements of the NIST quality metric.

For the CRAAC assessment, a CRAAC prototype has been built, experiments have been conducted, and results are reported in the next two chapters.

5.4.1 Performance Evaluation

As outlined in Chapter 4, CRAAC is more complex than the traditional RBAC model. It is anticipated that a CRAAC implementation would add more overheads due to its new LoA-linked access control decision-making, hence introducing additional performance costs. The level of the performance cost should be investigated comprehensively as an indicator of the CRAAC efficacy. For this purpose, we measure the performance of CRAAC in terms of the average access delay (AAD).

Definition 7 *AAD represents an average access latency a subject has to wait before the access request is processed. It is the difference between the time when an access request is sent and the time when an access control decision is made.*

To measure AADs under different configurations, experiments are to be designed. To obtain experimental results with high statistical significance, a large number of iterations have to be used to eliminate arbitrariness. In other words,

Table 5.2: NIST Access Control Quality Metric [4]

Element	**Description**
User Management	This describes the steps required for assigning and disassigning user capabilities into the access control system
Object Management	This describes the steps required for assigning and disassigning object access control entries into the access control system
Least Privilege	To what degree an access control system supports the least-privilege concept
Access Control Policy	This element is concerned about 1)the number of relationships required to create an access control policy and 2) the capabilities of policy encapsulation for policy combination, composition and constraint.
SoD support	Is an access control service support SoD? This is significant to prevent unintended accesses
Implementation and Evolution	This describes the degree to which an access control system is adaptable to the implementation and evolution of access control policies.
Horizontal Scope	What is the scope of coverage across platforms and applications an access control system can support? This can be restated as: is an access control system applicable for a specific application domain or and is it able to cater for a wide range of application domains?
Vertical Scope	This is concerned about the level of integration between an access control system and other systems, such as a database management system and operation system.
Safety	This describes the capability of an access control system to enforce safety. This is measured by the number of safety constraints an access control system can support.
Access Control Management	This describes the degree of freedom for access control management. It addresses the need to support different points of views for managing an access control system.
Performance	This describes the cost of running an access control system in terms of, for example, the number of operations required to grant/deny a user access to a certain resource object.
Conflict Resolution	Can an access control system resolve conflicts in access control policies?
Flexibility	This describes the level of flexibility in configuring an access control system. This may measure how modular an access control system is to support an external component.

to ensure statistical significance, the number of iterations (denoted as n), over which the AAD is measured, needs to be determined. The larger the value of n, the less the effect of arbitrariness on the AADs. To determine n, an experiment is conducted by setting n to different values while measuring the AADs in milliseconds. A certain value of n, at which the arbitrariness tends to disappear, is selected and used in all subsequent experiments. This will be shown in Section 6.5.

5.4.2 Security Evaluation

Safety is important in any access control system. An access control system is said to be safe if no permissions are leaked to an unauthorised entity [4]. In fact, this definition of safety does not address a case where a legitimate user is given a permission inadvertently. For example, if an access control system grants a legitimate teller, at a bank counter, a signature authorisation capability at the same time, thus the access control system is not safe. Therefore, this definition of safety is incomplete. In this thesis, safety of an access control configuration is defined as follows:

Definition 8 *The safety of an access control system refers to a system state where no permission is leaked to an unauthorised entity nor to a legitimate entity unintentionally.*

Theoretically, safety is proven undecidable [79]. In fact, safety is achieved by using a limited access control models or via constraints [80, 4]. An access control system should support safety by other practical mechanisms such as constraints. Although this means that there is no automatic proof of safety. It is still possible to give a manual proof of safety of an access control model [81, 82, 37]. In general,

SSD and DSD are the practical ways to enforce safety of an access control configuration. Since CRAAC is built on the RBAC model, CRAAC supports safety by utilising both SSD and DSD constraints. Thus CRAAC should be at least as safe as the RBAC model in the same access control configuration. In fact, it is assumed that the complexity of the CRAAC will not impose security loophole or reduces the overall system security. Moreover, the contextual information along with the provider are part of the TCB. In fact, CRAAC safety regarding DoS attacks will be investigated in Section 6.5.7 later on.

CRAAC further enforces safety by introducing the LoA constraints. Thus, no permission is granted to a subject unless the subject has satisfied certain LoA requirements. In fact, the LoA constraints provide a higher level of safety than that of the RBAC model. For example, in the RBAC system, Eve can break into the system by only compromising Bob's login credential (e.g. user-name and password), whereas in CRAAC, she has to compromise more access control barriers, such as being in a certain known secure location and/or possessing other stronger authentication credentials. By using the SSD, DSD, and LoA constraints, CRAAC should be able to eliminate the possibility of permission leakage to an unauthorised entity and unintended permissions to a legitimate entity.

By expressing an access control policy in terms of LoA instead of the contextual attributes, CRAAC reduces the effect of compromising policy stores on the overall system security. In a CAAC-based access control policy, compromising the policy store provides an attacker a comprehensive knowledge of the system and what contextual constraints he/she needs to satisfy in order to gain access to a particular resource object. However, in CRAAC, even if the policy store was compromised, the attacker will not know which contextual attributes he/she has to compromise to provide the required OLoA for the resource object. In addition,

he/she will not even know the set of contextual attributes[5] a legitimate subject has subscribed to. This increases the overall security of the CRAAC model.

5.4.3 CRAAC Evaluation TestBed

As mentioned earlier, the CRAAC prototype has been built as a proof of concept and as a platform to conduct performance assessments and security evaluation. The prototype is hosted on Ubuntu 9.04 operating system on a DELL desktop with 2x Intel(R) Pentium(R) 4 CPU 3.00GHz processors with 1017MB memory. The prototype is implemented as a Java desktop application with Java^{TM}SE Runtime Environment build 1.6.0_13b03. It uses MySQL 5.0.75 to store UA (i.e. both static and dynamic) and high-level contextual attribute values (i.e. emulation). This part is assumed to be part of the TCB, since it deals with contextual information and providers. The prototype uses JbossXACML 2.0.3 [83] for PA policy evaluation. In fact, this corresponds to the traditional RBAC authorisation engine. Other CRAAC policies (i.e. the resources' OLoA requirements, the RLoA derivation methods, and system configuration parameters) are written in XML 1.0 and are parsed using SAXParser [84].

5.5 Chapter Summary

This chapter has discussed the motivations and design preliminaries of the CRAAC architecture. The chapter has also highlighted the policy types and policy retrieval modes supported by CRAAC. In addition, CRAAC evaluation has been discussed. CRAAC is said to be at least as safe as RBAC, since it uses practical tools such as SSD, DSD, and LoA constraints to control access to protected resource objects. The next two chapters will study the CRAAC model in two

[5] In case of the RLoA-only mode

modes of working (i.e. RLoA-only and AttributeLoA-only mode) along with an investigation of their performance.

Chapter 6

The RLoA-only Mode

6.1 Chapter Introduction

There is a need for an architecture to realise the LoA-aware access control vision discussed in the previous chapters. The architecture should support the four modes of working[1] proposed by CRAAC. This chapter describes the design of the CRAAC architecture along with its components. It also shows in detail how the RLoA-only mode uses the architecture to support its services (i.e. controlling access to resources using RLoA). In addition, this chapter investigates and evaluates the performance of the RLoA-only mode against that of the basic-RBAC mode.

The remaining part of this chapter is structured as follows: -

- Section 6.2 introduces the CRAAC architecture.
- Section 6.3 discusses the design of the architecture in detail covering its fundamental services and components.
- Section 6.4 shows how the RLoA-only mode uses the CRAAC architecture

[1]See Section 4.6 for more detail

to control access to resource objects using RLoA.

- Section 6.5 investigates the performance of the RLoA-only mode and compare it against the performance of the basic-RBAC mode.
- Section 6.6 summarises the chapter.

6.2 The Architecture Overview

CRAAC classifies its services into three major functional blocks: Access Control Infrastructure (ACI), Context Management Infrastructure (CMI), and LoA Derivation Infrastructure (LoADI). The LoADI is responsible for calculating users' LoA/RLoA value(s) based upon the latest contextual information fed from the CMI. It feeds those values to the ACI in order to make access control decisions for the corresponding user. This design approach separates the functions of access control from that of context management by introducing a layer of abstraction (i.e. LoADI) to loosely bridge both infrastructures. This approach provides a high degree of separation of duties among the major functional blocks and a high level of functional encapsulation, which leads to flexibility and extensibility of the overall architecture. This may also add another level of complexity specially when the CRAAC is deployed as a distributed service. Efforts have been made to stream-line the inter-component communication to reduce the number of interactions and communication overheads.

The major functional blocks of the CRAAC architecture are outlined in Figure 6.1, and the RLoA-only mode inter-component communication is illustrated in Figure 6.2. The following section gives a detailed description of the three CRAAC infrastructures with an emphasis on those architectural components that are used in the RLoA-only mode.

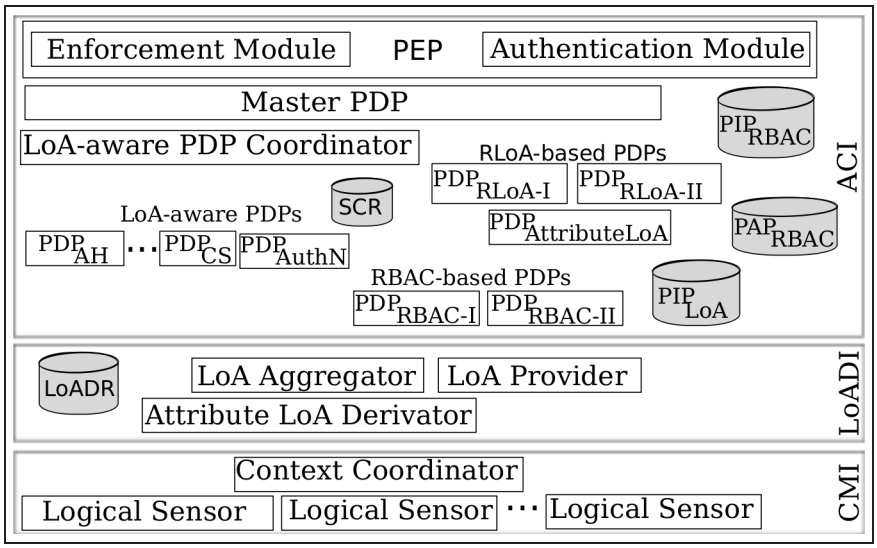


Figure 6.1: CRAAC Architectural Components

6.3 The Architecture in Detail

6.3.1 Access Control Infrastructure (ACI)

The ACI encompasses components for both authentication and authorisation. CRAAC does not specify or require a specific authentication service. Rather, any existing authentication service can be plugged into the architecture. The CRAAC authorisation service is built upon the traditional RBAC model and uses standard access control components like Policy Enforcement Point (PEP) and Policy Decision Point (PDP). CRAAC, yet, adds additional components such as Master PDP, LoA-aware PDPs and their Coordinator, and RLoA-based PDPs to provide the novel CRAAC authorisation services.

Generally, the fundamental role of a PEP is to enforce an access control decision made by a PDP. In CRAAC, PEP receives an access request from a subject, extracts the attribute values contained in the access request, packages these values along with the attribute names in a standard XACML 2.0 request context [77], and forwards them to the Master PDP for evaluation. Then, it enforces the

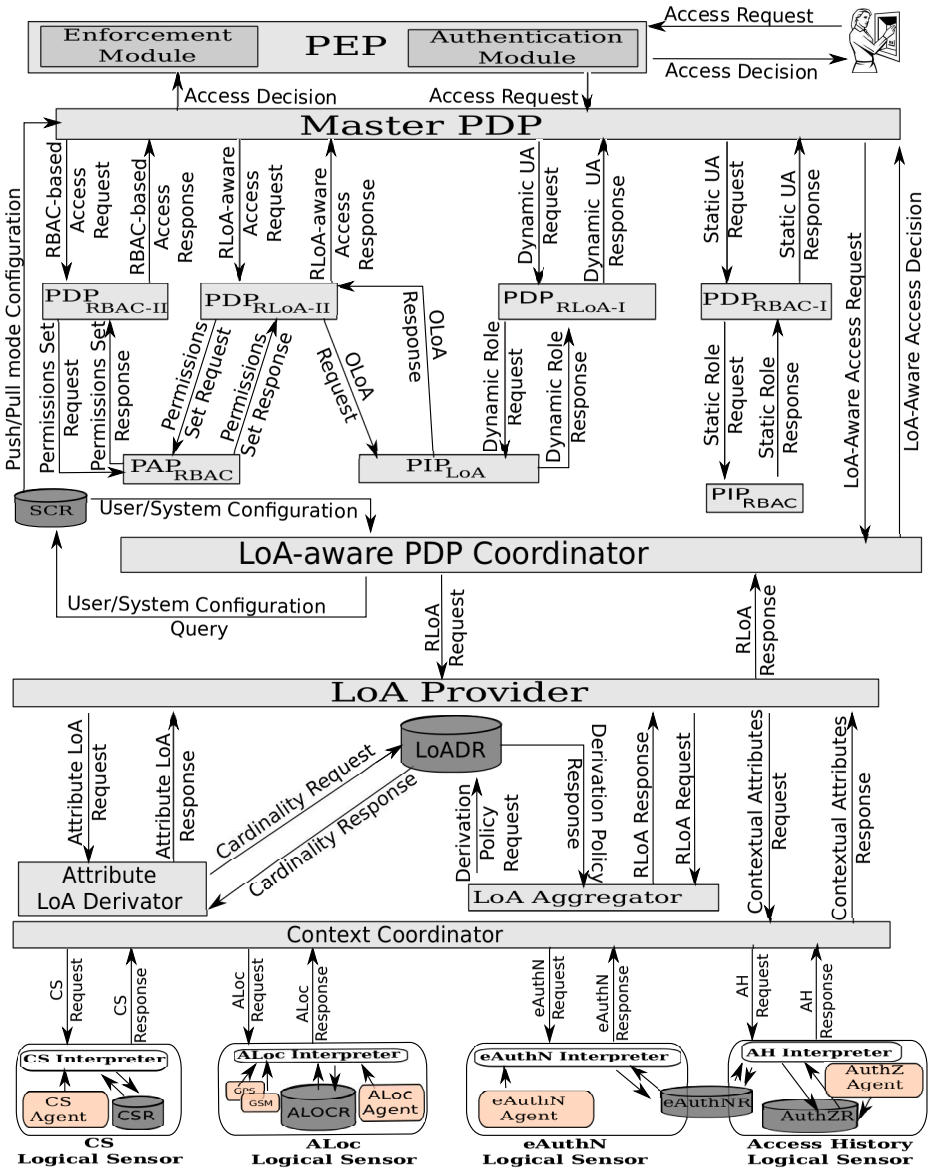


Figure 6.2: The CRAAC Architecture: the RLoA-only Mode

access control decision made by the Master PDP. Moreover, PEP assists the *access history logical sensor* and the *eAuthN logical sensor* to monitor a requester's access history by notifying them the authentication and authorisation outcomes associated to the requester.

The Master PDP is responsible for orchestrating CRAAC different types of PDPs to render a final access control decision. It also should resolve conflicts,

if any. CRAAC supports three types of PDPs: RBAC-based, LoA-aware, and RLoA-based PDPs.

The RBAC-based PDPs are used to perform the traditional RBAC functions such as UA and PA. PDP_{RBAC-I} is dedicated for the former function and $\text{PDP}_{RBAC-II}$ is responsible for the latter function. These PDPs are fundamental in any CRAAC mode of working.

The LoA-aware PDPs and their Coordinator are used to support an application scenario where individual contextual attributes' LoA values are used to grant/deny access to resource objects (i.e. the AttributeLoA-only and Combined modes). Thus, for each contextual attribute, a separate LoA-aware PDP is required. CRAAC currently uses three LoA-aware PDPs: PDP_{CS}, PDP_{AuthN}, and PDP_{AH}. PDP_{CS} is a decision engine where decisions are made based upon requesters' levels of assurance in their channel security attribute. PDP_{AH} is a decision point that supports history-aware access control. Finally, PDP_{AuthN} is a dedicated authorisation decision engine that supports access control policies based on both eToken and ALoc attributes.

Since the LoA-aware PDPs operate on LoA values calculated at the LoADI, there is a need for a service that acquires the LoA values from the LoADI. There are two design alternatives for this service. One is to let each LoA-aware PDP to directly communicate with the LoADI. This option may impose a high level of communication overhead on the LoADI. This is because a requester may subscribe to multiple contextual attributes, then multiple LoA-aware PDPs will be involved each of which will have to establish an independent communication link with the LoADI. Excessive communication burden on the LoADI may increase access delays slowing down the performance and reducing the responsiveness of the CRAAC model. In addition, this design option violates the high functional encapsulation requirement that CRAAC aims at achieving. In fact, any change

in the number of the LoA-aware PDPs will be visible to the LoADI. Therefore, CRAAC adopts another design option, where a LoA-aware PDP Coordinator is introduced to coordinate the communications between the LoA-aware PDPs and the LoADI and between the LoA-aware PDPs and the Master PDP.

The third class of PDPs is the RLoA-based PDPs that include PDP_{RLoA-I} and $\text{PDP}_{RLoA-II}$. These PDPs perform the functions of role and permission adaptation based upon RLoA. When making an access request, a user is, firstly, assigned a static role by PDP_{RBAC-I} using the user's static attribute(s). The static role may be dynamically adjusted to another role if the user's RLoA satisfies a certain threshold. This RLoA-linked role adjustment is performed by PDP_{RLoA-I}. Similarly, $\text{PDP}_{RLoA-II}$ adaptively adjusts access permissions based on the user's RLoA value. It is worth emphasising that the RLoA-based PDPs are only used in both RLoA-only and Combined modes. Another PDP that is relevant to this context, the $\text{PDP}_{attributeLoA}$. The $\text{PDP}_{attributeLoA}$ is a decision point that performs the same function as PDP_{RLoA-I}, but in the AttributeLoA-only mode. It generates a user's dynamic role based on the user's individual contextual attribute LoA values instead of the an RLoA value as in the RLoA-only mode.

When an access request is received, the Master PDP needs to be aware of the corresponding PDPs that should be used for the request. In fact, there is a need for a component to store such information. The System Configuration Repository (SCR) is, hence, used for this purpose. The SCR also contains protocol identifiers to indicate the type of the communication protocol each PDP uses (i.e. currently CRAAC supports XACML 2.0 request/response context).

6.3.2 LoA Derivation Infrastructure (LoADI)

The main functions performed by the LoADI are to derive LoA and RLoA values based on the requester's contextual information and to feed the values to the ACI. It has three main components: the LoA Provider, Attribute LoA Derivator and LoA Aggregator. The Attribute LoA Derivator derives a LoA value for a given contextual attribute. The composite effect on a user's LoA, when multiple attributes are involved, are assessed by the LoA Aggregator. Once LoA/RLoA values are calculated, the LoA Provider is responsible for sending them to the ACI for access control decision-making. Basically, the LoADI plays an intermediary role bridging both ACI and CMI.

The Attribute LoA Derivator receives the contextual attribute values (i.e. in terms of LoA ranks) from the LoA Provider. It then calculates the corresponding LoA values (i.e. L2WC) before sending the LoA values back to the LoA Provider.

The LoA Aggregator is responsible for aggregating all LoA values into one RLoA value, which will be used to control access to resource objects[2]. This component receives LoA values computed by the Attribute LoA Derivator. Then, it queries the LoADR for the possible aggregation method (i.e. *Weakest-link* or *Elevating*) to use before aggregating the LoA values into one RLoA value. Finally, it sends the RLoA value back to the LoA Provider.

6.3.3 Context Management Infrastructure (CMI)

A context-aware application may interface a diverse range of devices such as sensors, software applications, and other context-aware components. CRAAC uses the notion of logical sensors [85] to refer to all devices, entities, or software components that sense or provide contextual data. Examples of logical sensors currently

[2]In the RLoA-only mode

supported by CRAAC include those for monitoring channel security levels, sensing eToken assurance levels, detecting access locations, and tracking users' access history. In fact, CRAAC extends the definition of logical sensors to include an interpretation service that generalises high level contextual information from the row contextual data acquired from the sensors. For example, an interpretation service may receive location information in terms of zones and, then, maps the zones into the corresponding LoA (i.e. ranks). It is worth emphasising that the interpretation function may vary depending on the type of the contextual attribute and the representation of its value. This is why an interpretation service is encapsulated inside a logical sensor, instead of using a central interpretation service for all sensors.

CRAAC introduces the Context Coordinator in order to provide an interface for the LoADI to access the contextual information from the logical sensors transparently. Indeed, the Context Coordinator hides the design, implementation, and configuration details of logical sensors from the LoADI. As this thesis focuses on the use of the contextual information for access control, the component design of the CMI is beyond the scope.

6.4 RLoA-only Mode Data-Flow

In this section, a detailed illustration of the steps and messages exchanged in the RLoA-only mode is given through a case study. The case study assumes a legitimate user, Bob, is trying to get a "Write" access on the "srv.config" file object. Figure 6.3 illustrates the data-flow of this access request using the RLoA-only mode.

Given Figures 6.2, the following describes both steps and message exchange required to grant/deny Bob's access to the requested resource object:

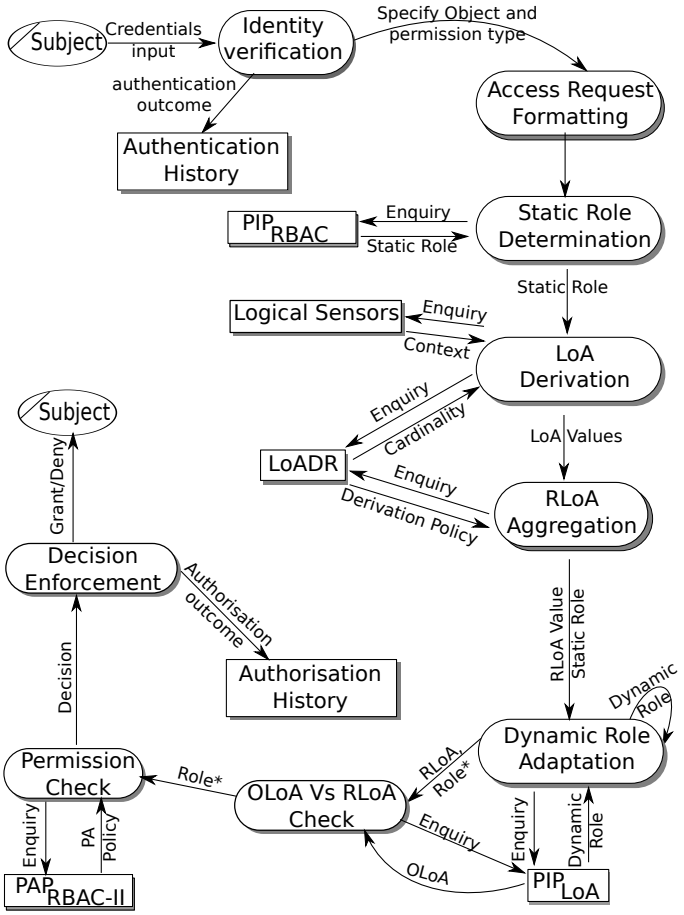


Figure 6.3: Data-flow in the RLoA-only Mode

1. Bob sends an access request to PEP for a *"Write"* permission on the *"srv.config"* file object. Before that, an authentication phase should take place in order to verify the identity of Bob. The outcome of the authentication phase is recorded in the eAuthN logical sensor's repository. If the authentication fails, the access request is denied, otherwise proceed to next step.

2. PEP forwards the access request to the Master PDP for an authorisation decision.

3. The Master PDP sends the subject ID (i.e. Bob) to PDP_{RBAC-I} that

determines Bob's static role using the traditional RBAC model (i.e. UA function).

4. PDP_{RBAC-I} queries PIP_{RBAC} for Bob's static role.

5. PIP_{RBAC-I} returns the static role of Bob (i.e. Staff).

6. PDP_{RBAC-I} forwards Bob's static role, Staff, to the Master PDP.

7. The Master PDP sends a context-aware access request to the LoA-aware PDP Coordinator. In fact, the Coordinator will not consult any subordinate LoA-aware PDPs as no access control decision is expected from the LoA-aware PDPs in this mode of working. Instead, the LoA-aware PDP Coordinator initialises the LoADI in order to start LoA/RLoA derivation for Bob.

8. The LoA-aware PDP Coordinator queries the SCR for the current system settings. The system settings include the context-awareness flag which indicates the state of the context awareness (i.e. enabled/disabled) and the mode the system is configured to use. In addition, it queries the SCR for the types of contextual attributes that Bob has subscribed to.

9. The SCR sends the current settings to the LoA-Aware PDP Coordinator. For instance, the context awareness flag is enabled, the system is configured to use the RLoA-only mode and the list of contextual attributes that Bob has subscribed to is {eToken, ALoc, AH, CS}.

10. The LoA-Aware PDP Coordinator sends an RLoA request to the LoA Provider passing it, as an argument, the types of the contextual attributes Bob has subscribed to. In fact, this step transfers the control to the LoADI. The LoA Provider acts as an interface to the LoADI. It receives requests

for RLoA derivation and forwards the RLoA value back to the LoA-aware PDP Coordinator.

11. The LoA Provider sends a request to the Context Coordinator to retrieve Bob's up-to-date contextual attributes values. The Context Coordinator consults its logical sensors for such values. Once the values are ready, it sends them back to the LoA Provider.

12. The LoA Provider forwards the attribute values to the Attribute LoA Derivator in order to derive the corresponding LoA values (i.e. L2WC). To perform L2WC, the Attribute LoA Derivator consults the Level of Assurance Derivation Repository (LoADR) for the corresponding contextual attribute cardinalities.

13. Once the attribute LoA values are calculated by the Attribute LoA Derivator, they are sent back to the LoA Provider, which forwards them to the LoA Aggregator.

14. The LoA Aggregator consults the LoADR for the aggregation method (e.g. *Weakest-link* or *Elevating*) that should be used for generating the RLoA value from the set of the attribute LoA values received from the LoA Provider. Once the RLoA value is calculated, the LoA Aggregator forwards it to the LoA Provider, which sends it back to the LoA-aware PDP Coordinator that transfers it back to the Master PDP.

15. There is a need to check if Bob's static role should be adjusted (i.e. dynamic UA) based on the received RLoA value. Thus, the Master PDP consults the PDP_{RLoA-I} that checks the PIP_{LoA} for a role adaptation. The new role (i.e. dynamic role), if any, will be sent back to the Master PDP.

16. The Master PDP forwards Bob's access request to the $PDP_{RLoA-II}$. The request contains Bob's RLoA value, role (i.e. either the static or dynamic one), the object to be accessed (i.e. *srv.config*), and the access mode (i.e. *Write*). The access request is encapsulated in an XACML 2.0 request context object.

17. The $PDP_{RLoA-II}$ then performs the following operations to reach an access control decision based on the attribute values received:

    (a) Checks if Bob's RLoA value is greater than or equal to the OLoA value required by *srv.config*. To do this, it queries the PIP_{LoA} for the *"srv.config"* OLoA requirement. If the check succeeds, it performs the next step, otherwise a *deny* decision is generated.

    (b) Checks whether Bob's current role is permitted to have a *"Write"* access on *"srv.config"*. If this check fails, a *deny* decision is generated, otherwise, a *grant* decision is made.

    (c) The generated decision (i.e. *grant/deny*) is sent to the Master PDP.

18. The Master PDP forwards the access control decision received from the $PDP_{RLoA-II}$ to the PEP for actual enforcement.

19. It is worth emphasising that the authorisation result (i.e. *grant/deny*) is recorded in the access history logical sensors' local repository in order to keep Bob's access history up-to-date.

6.5 RLoA-only Mode Performance Evaluation

This section investigates and compares the performance of the RLoA-only mode and the basic-RBAC mode. The performance is measured in terms of AADs

using experiments. The effect of the following factors is investigated: different PA policy sizes, *pull* and *push* policy retrieval modes, the number of enabled roles of an access requester, the RLoA value, and the queuing delays.

To ensure statistical significance in the experiments, the number of iterations (denoted as n) over which an AAD is measured should be determined. An experiment has been conducted by setting n to different values while measuring the AADs in milliseconds. Figure 6.4 shows the results of this experiment that indicates the bigger the value of n, the less effect the other initialisation factors have on the AADs. The trend of the graph shows an n value higher than 2.8k (i.e. 2800 iterations) is sufficient, thus all subsequent experiments use an n value of 3000.

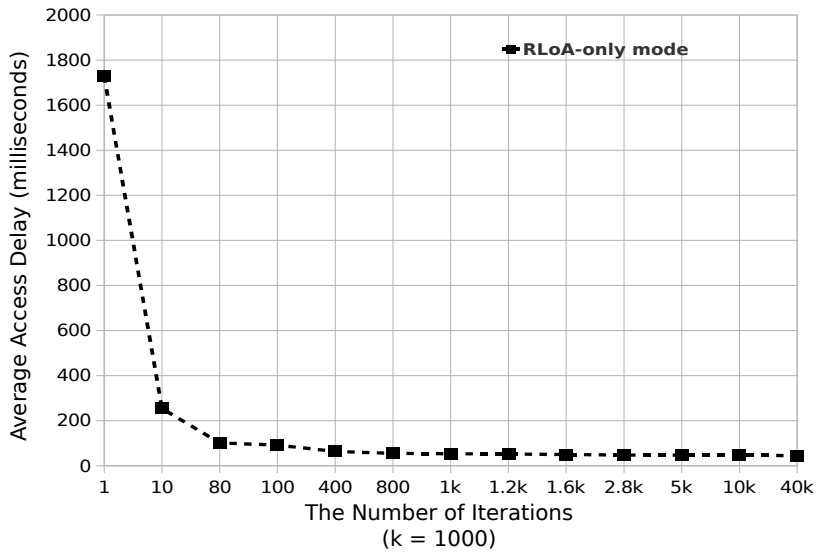


Figure 6.4: The Number of Iterations Determination

6.5.1 The Effect of the PA Policy Size

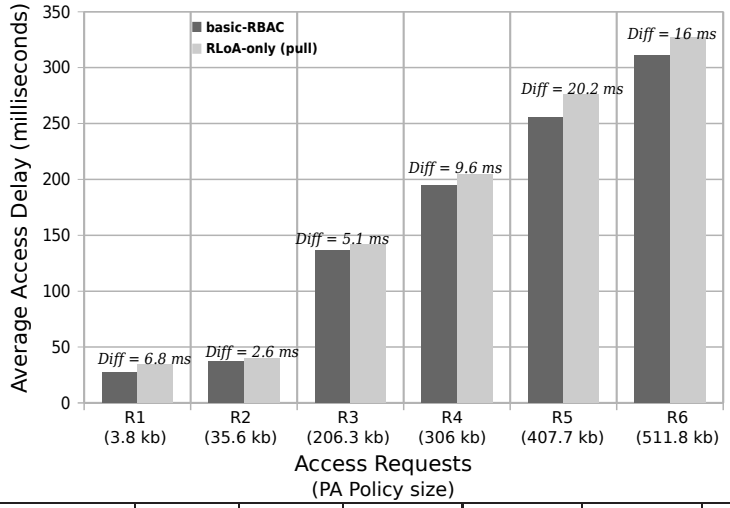
This experiment investigates the effect of the PA policy size on the AAD in the RLoA-only mode. It assumes a user, Bob, with one enabled role, Staff, makes six access requests to six different resource objects. The six access requests are

denoted as R1 through R6 with PA policy sizes of 3.8, 35.6, 206.3, 306.0, 407.7, and 511.8 KBytes, respectively. Moreover, the experiment assumes that the PA policies only use *permit* rules and the algorithm used to combine the rules is *deny-override* [77]. The experiment also assumes Bob's RLoA value is always greater than or equal to the required OLoA values of the requested resource objects across the six cases. The latter assumption ensures the execution of the RLoA-only mode always reaches the stage where the PA policy files are opened. Because if the $RLoA < OLoA$, a deny decision will be generated immediately without checking the PA policy (Step 17 in Section 6.4). The former assumption enforces every rule contained in the PA policy to be evaluated before making an access control decision. Thus, the experiment precisely captures the effect of the PA policy size on the AAD.

Prior to the experiment, two main results are anticipated. Firstly, the larger the PA policy, the longer it takes to evaluate an access request in both RLoA-only and basic-RBAC modes. This is because the larger the XACML policy file, the longer it takes for the system to parse the policy and evaluate the contained rules. Secondly, the amount of overhead introduced by the RLoA-only mode additional functions should be consistent and should be almost the same across the 6 policy sizes. In other words, if the RLoA-only mode adds Δt_i to the $AAD_{basic-RBAC}$ in request i, then it should add Δt_j to request j, where $i \neq j$ and $\Delta t_i \approx \Delta t_j$. This is because, the total access delay for the RLoA-only mode is: $AAD_{RLoA-only} = AAD_{RLoAcalculation} + AAD_{OLoAcheck} + AAD_{basic-RBAC}$, where $AAD_{RLoAcalculation}$ is the time taken to calculate RLoA, $AAD_{OLoAcheck}$ is the time taken to evaluate the requester's RLoA against the requested object's OLoA, and $AAD_{basic-RBAC}$ is the time taken for user-role mapping (i.e. UA function) and role-permission determination (i.e. PA function). The difference of AADs in the RLoA-only and basic-RBAC modes (i.e. Δt) is $(AAD_{RLoAcalculation} + AAD_{OLoAcheck})$, which is

independent of the PA policy size.

Figure 6.5 shows the results of the experiment. The figure shows two sets of AADs measured in milliseconds against the six different access requests. One set is from the basic-RBAC mode and the other is from the RLoA-only mode. Three observations can be made from the figure. Firstly, the AADs in both modes increase steadily as the PA policy size increases. This observation is in line with our expectation.



	R1	R2	R3	R4	R5	R6
basic-RBAC	27.45	37.46	136.70	194.81	255.51	310.64
RLoA-only$_{pull}$	34.27	40.07	141.76	204.37	275.72	326.68

Figure 6.5: The PA Policy Size Effect: RLoA-only$_{pull}$ vs basic-RBAC

Secondly, the average AAD introduced by the additional RLoA-only mode functions across the six access requests is about 6.26% of the AAD taken in the basic-RBAC mode. This observation indicates that in RLoA-only mode, the majority part of the AAD is caused by the basic RBAC functions. The execution of the additional CRAAC functions in this mode only contributes a small part towards the total average access delay.

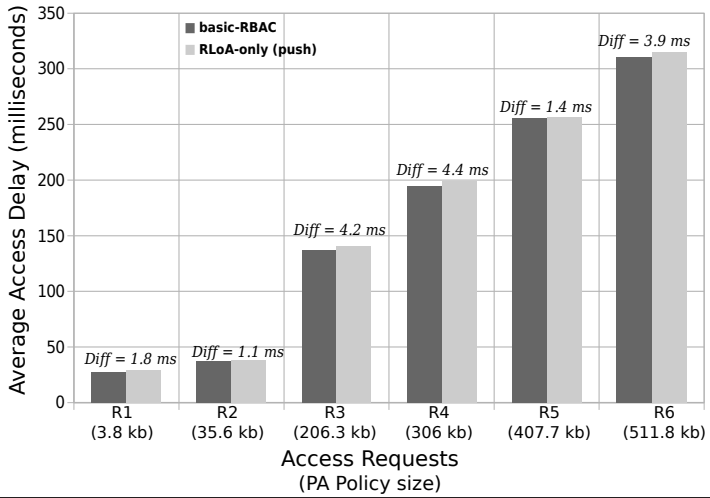
Thirdly, the AAD difference between the RLoA-only mode and basic-RBAC mode fluctuates across the 6 access requests. The experimental results show the

difference values for R1 through R6 fluctuate between 2.6 to 20.2 milliseconds and the difference values for R1, R2, and R3 are smaller than that for the other requests. After in-depth examinations and analysis of the RLoA-only mode operations and the experimental results, the following can explain this unexpected finding. The two additional functions performed by the RLoA-only mode are: (1) the dynamic role adaptation, which includes tasks of RLoA derivation and role adjustment, (2) the comparison of a user's RLoA value against the required resource's OLoA value. The former function introduces a constant time overhead. This is because dynamic roles are stored in a relational database and the time taken to retrieve a new role from the database should be almost the same across the six access requests. In addition, the LoA calculation and RLoA aggregation are expected to be the same for the same user on the same set of contextual attributes. However, the latter function (i.e. point (2)) introduces a variable time overhead. This is because the latter function uses the resources' OLoA policy file, which is written in XML[3]. Hence, CRAAC needs to parse this file to get the required OLoA value. The time it takes to get the OLoA value varies depending on the relative location of the requested resource object in the file. For instance, the time it takes to get the OLoA value of the first resource object in the resources' OLoA policy file is considerably less than the time taken to get the OLoA value of the last resource object stored in the same file. To summarise, accessing the resources' OLoA policy file introduces a variable non-negligible level of access delay, which depends on the relative position of the requested object in the file. This observation has motivated us to investigate how to reduce the variability in AAD. For this reason, the use of *pull* mode versus *push* mode for the resources' OLoA policy file has been evaluated.

[3]See Section 5.3 for more detail

6.5.2 The Effect of Resources' OLoA Policy Size: Push Vs Pull

One way to reduce the access delays introduced by the RLoA-only mode additional functionality is, perhaps, to have the resources' OLoA policy pushed into the system when the system initialises. Thus, at run time when an access request is received, CRAAC can do a memory access instead of a disk access (i.e. instead of opening and parsing the resources' OLoA XML policy file) in order to get the required OLoA value. Since memory operations take significantly less time than disk operations, using the *push* mode is expected to cut down the AADs, thus providing a better performance. Based upon this belief, a further experiment is conducted to investigate the level of performance enhancement when using the *push* mode for accessing the resources' OLoA policies. The results are plotted in Figure 6.6.



	R1	R2	R3	R4	R5	R6
basic-RBAC	27.45	37.46	136.70	194.81	255.51	310.64
RLoA-only$_{push}$	29.21	38.60	140.94	199.19	256.95	314.56

Figure 6.6: The PA Policy Size Effect: RLoA-only$_{push}$ Vs basic-RBAC

The following observations can be seen from Figure 6.6. Firstly, the overhead

incurred in accessing and parsing the resources' OLoA policy is the major contributor to the AAD difference between the two modes. In other words, $AAD_{OLoAcheck}$ plays a predominant role in the total value of $(AAD_{RLoAcalculation}+AAD_{OLoAcheck})$. When using the *push* mode, the $AAD_{OLoAcheck}$ value is reduced exhibiting approximately the same value of $(AAD_{RLoAcalculation} + AAD_{OLoAcheck})$ across the six access requests. Secondly, the *push* mode makes the RLoA-only mode marginally more expensive than the basic-RBAC mode in terms of AADs. For example, the $AAD_{RLoA-only_{pull}}$ for R2 was about 6.97% higher than $AAD_{basic-RBAC}$, but when the *push* mode is used, this figure is reduced to about 3.04%. Averaged over the six access requests, the figure is reduced from about 6.26% in the *pull* mode to 1.75% in the *push* mode.

As a conclusion, the performance of the RLoA-only mode is comparable to that of the basic-RBAC mode and the difference between the two modes is almost negligible when the *push* mode is used for accessing the resources' OLoA policy.

6.5.3 The Effect of the Number of Enabled Roles

In real-life situations, a subject may hold multiple organisational roles. For example, a lecturer may also be in charge of the admission task within the school. An experiment is conducted to evaluate the performance of the RLoA-only mode when a multiple number of enabled roles (i.e. 1, 2, 4, 10, and 30 enabled roles) are assigned to a subject (e.g. Bob). The experiment assumes all Bob's access requests are on the same resource object. In addition, it uses the *push* mode to access the resources' OLoA policy file. Figure 6.7 shows the results of the experiment.

The general trend of the graph in Figure 6.7 indicates that the AAD increases linearly as the number of enabled roles increases. The more enabled roles Bob

	1	2	4	10	30
RLoA-only$_{push}$	50.29	50.94	52.14	53.69	54.49

Figure 6.7: RLoA-only$_{push}$: the Effect of the Number of Enabled Roles

holds, the longer it takes to process his access requests. For example, when Bob holds only one role, the AAD is about 50.3 milliseconds and when he holds 30 roles, the AAD is about 54.5 milliseconds. This 8.4% increase in AAD (e.g. as the number of roles increases from 1 to 30) is not as significant as in the case where the size of the PA policy increases. For example, when the size of the PA policy increases only by about 9 times[4], the AAD increases by about 32%. In fact, the marginal increase in AAD, as the result of increasing the number of enabled roles, is due to the fact that the operations required here are memory operations not disk operations. In other words, once the corresponding PA policy is processed (i.e. either in push or pull mode), the rules contained in the policy will be compared against the set of enabled roles of the access requester. The comparison does not require any further policy manipulations (i.e. disk operations).

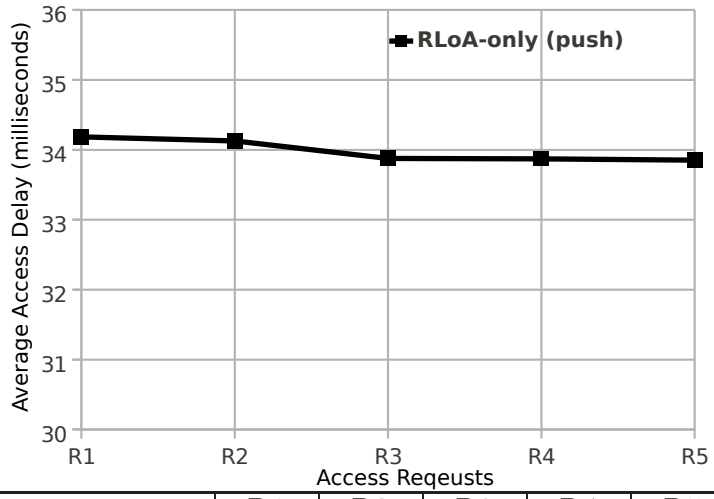
6.5.4 The Effect of the Attribute LoA Derivator

In this experiment, the effect of the Attribute LoA Derivator is evaluated. In fact, this experiment assesses the cost of the L2WC method (i.e. ROC) on the AAD,

[4]From 3.8 to 35.6 KByte in Figure 6.6

since the fundamental function of the Attribute LoA Derivator is to compute the LoA values for the contextual attributes given their corresponding ranks. The experiment runs 5 access requests R1 through R5 made by an access requester (i.e. Bob). To emphasis the effect of the Attribute LoA Derivator, Bob is assumed to request the same resource object in all the 5 runs. This eliminates the effect of the PA policy size on the AADs. In addition, Bob is subscribed to a set of contextual attributes, namely ALoc, eToken, CS and AH. The experiment uses the ALoc attribute to evaluate the effect of the L2WC. Thus, changing the ALoc rank in every access request should capture the effect of the L2WC on the AAD. For this purpose, the experiment assumes the access location space is divided into 1000 zones, where zone_1 is the most secured zone and zone_{1000} is the least secured one. The 1000 zones are, in fact, the ALoc attribute cardinality. It is also assumed that each access request is made from a different zone (i.e. different ALoc rank). This means that after each access session, Bob roams to a new zone before starting a new access session. That is, when receiving R1, R2, R3, R4, and R5, the ALoc_{Bob} will be $zone_1$, $zone_{100}$, $zone_{500}$, $zone_{800}$, and $zone_{1000}$, respectively. The results of this experiment is depicted in Figure 6.8.

The figure shows the AADs in milliseconds for 5 access requests made by Bob on the same resource object. Each access request varies in ALoc value (i.e. 1, 100, 500, 800, and 1000), but operates on the same ALoc attribute cardinality (i.e. 1000). As can be seen from the graph, the $AAD_{R1} > AAD_{R2} > AAD_{R3} > AAD_{R4} > AAD_{R5}$. In other words, the more secure the zone is, the longer it takes to reach an access control decision. This is understandable, since the *ROCs* will require more iterations to compute the corresponding LoA value. However, the overhead cost introduced by the LoA derivation process is insignificant. The AAD_{R1} is about 1% higher than that of AAD_{R5}, which is negligible. This confirms our observation made in Subsection 4.4.3 that *ROC* is a lightweight algorithm



	R1	R2	R3	R4	R5
RLoA-only$_{push}$	34.18	34.13	33.88	33.87	33.85

Figure 6.8: RLoA-only$_{push}$: the Effect of the Attribute LoA Derivator

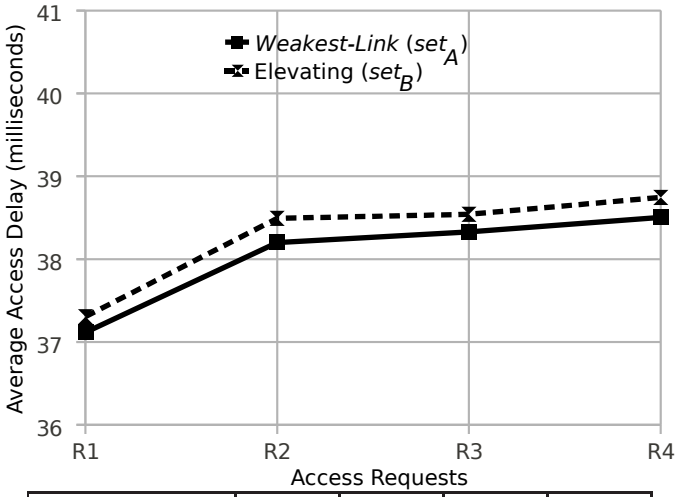
suitable for UbiComp environments.

6.5.5 The Effect of the LoA Aggregator

As discussed in Chapter 4, two RLoA aggregation methods are used: *Weakest-link* and *Elevating*. This experiment investigates the run-time costs of the two methods. The experiment runs 4 access requests made by Bob on the same resource object in two run sets. The first set assumes all Bob's contextual attributes are in a *Weakest-link* relationship and the second set assumes they are in an *Elevating* relationship. Each of the 4 access request is assumed to operate on a different set of contextual attributes. That is, in the first access request, Bob is subscribed to only one contextual attribute. In the second, third, and fourth access requests, Bob is subscribed to 2, 3, and 4 contextual attributes, respectively.

Figure 6.9 depicts the results of the experiment. It shows two sets of AADs measured in milliseconds against four different access requests. The first set (i.e set$_A$) represents a scenario where all the contextual attributes of Bob are in a *Weakest-link* relationship, whereas in the second set (i.e. set$_B$) the contextual

attributes are in an *Elevating* relationship. The following observations can be made from the figure.



	R1	R2	R3	R4
Weakest-link	37.12	38.20	38.33	38.50
Elevating	37.31	38.49	38.54	38.75

Figure 6.9: The Effect of the LoA Aggregator on the RLoA-only mode AADs

Firstly, in both sets, the more contextual attributes Bob is subscribed to, the longer it takes to reach an access control decision. This is because, as shown in Section 4.5, the more contextual attributes involved in RLoA aggregation, the longer time it takes the LoA Aggregator to perform the aggregation. If R1 is excluded, the AAD in both sets increases linearly. This behaviour confirms the linear (i.e. O(n)) the complexity of both aggregation methods.

Secondly, the AAD of set$_A$ is slightly shorter than that of set$_B$. This is because the *Weakest-link* security relationship (e.g. set$_A$) requires only boolean operations to find the minimum, which takes less time than the multiplications and subtractions required in case of the *Elevating* relationship.

Finally, it can be seen from Figure 6.9 that the effect of LoA Aggregator (i.e. in both sets) on the AAD in the RLoA-only mode is insignificant. This indicates the RLoA-only mode performs lightweight operations when aggregating the LoA

values into one RLoA value in both *Weakest-link* and *Elevating* relationships.

6.5.6 The Effect of the Queuing Delays

In all previous experiments, access delays as caused by queuing has not been studied. The previous experiments assume an access request is received after the previous one is completely processed. The main reason for this was to evaluate the effect of the individual architectural components on the overall AAD. However, access requests in a real-life scenario may need to be queued, since they may arrive at a rate higher than the rate the system takes to process an access request. As a result, users will experience a higher level of AADs. Obviously, the higher the access request arrival rate, the longer a subject would have to wait before his/her access request is processed, thus the longer the AADs.

The average access delay experienced by an access requester typically consists of two components: the queuing delay time (t_q) and the processing time (t_p). The former is the time an access request spends in the queue waiting to be processed, whereas the latter is the time the system takes to process the access request. In other words, $AAD = t_p + t_q$. The effect of t_p on AADs has been discussed in detail in the previous experiments. This experiment, however, evaluates the effect of t_q on the overall AAD.

The experiment tries to answer the following questions:

1. What is the turning point (i.e. threshold) beyond which the queue starts to build up?

   The threshold is the point beyond which the RLoA-only mode is expected to perform slowly (i.e. in terms of overall AADs). In case of excessive delay, there may be a need to add more authorisation servers to process access requests simultaneously. Thus, it is required to determine whether

the RLoA-only mode needs multiple servers. This is achieved by knowing the turning point.

2. Is t_q in RLoA-only mode comparable to the one of the basic-RBAC mode?

The following assumptions and system set-up are used in the experiment:

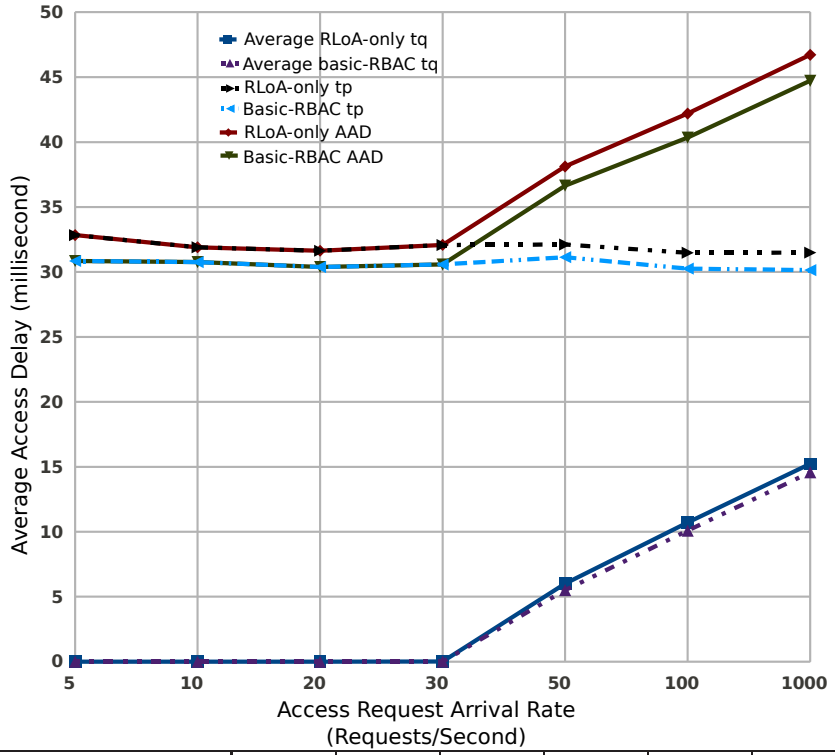
1. The experiment uses a first-come-first-served queue and process only one access request at a time. That is, all access requests are processed sequentially by one authorisation server.

2. All access requests are made on the same resource object. This assumption eliminates the effect of the PA policy size on the overall AADs, therefore emphasising the queuing delay effect on the AADs.

3. The *push* mode is used to access the resource's OLoA policy.

The experimental results are plotted in Figure 6.10. The figure shows two sets of AADs measured in milliseconds against seven different rates of access request arrival rates (i.e. 5, 10, 20, 30, 50, 100, and 1000 access request per second). One set is from the basic-RBAC mode and the other is from the RLoA-only mode.

As can be seen from Figure 6.10, the turning point beyond which the queue starts to build up is around the access rate of 30 access requests per second in both modes. The average access delay increases steadily when the arrival rate goes beyond this point. This is because, from this point onwards, the average interval between access request arrivals is shorter than the average request processing time. For example, when the arrival rate is 50 access request/second, the average interval between two requests is 20 ms, which is shorter than $t_p \approx 32.12$ ms in the RLoA-only modes. To minimise t_q, the system administrator may consider the use of multiple authorisation servers when arrival rates goes beyond 30 access



	5	10	20	30	50	100	1000
RLoA-only Average t_p	32.84	31.91	31.64	32.08	32.12	31.48	31.49
RLoA-only Average t_q	0.00	0.00	0.00	0.01	6.01	10.72	15.23
RLoA-only AAD	32.84	31.91	31.64	32.09	38.13	42.20	46.72
basic-RBAC Average t_p	30.85	30.76	30.39	30.58	31.14	30.26	30.15
basic-RBAC Average t_q	0.00	0.00	0.00	0.00	5.51	10.09	14.58
basic-RBAC AAD	30.85	30.76	30.39	30.58	36.65	40.35	44.73

Figure 6.10: RLoA-only Vs basic-RBAC: the Effect of Queuing Delay

requests/second. This possibility of using multiple authorisation servers needs further investigation and is considered in our future work.

6.5.7 The DoS Attack Resilience

The RLoA value plays an important role in the RLoA-only mode. If a user's RLoA is greater than or equal to the required resource object's OLoA, CRAAC will proceed to evaluate the corresponding PA policy. This policy evaluation introduces additional delays. On the contrary, if RLoA $<$ OLoA, CRAAC will

stop further processing. That is, it will not evaluate the request against the PA policy, generating a *deny* decision no matter what permissions may be assigned to the access requester. As a result, the AAD will be considerably less than the one endured when RLoA $\geq$ OLoA. This experiment is conducted to investigate the AADs under various RLoA value settings.

The experiment assumes the following:

1. All PA policy sizes are equal across all access requests (i.e. 10 access requests, R1 through R10, made by Bob).

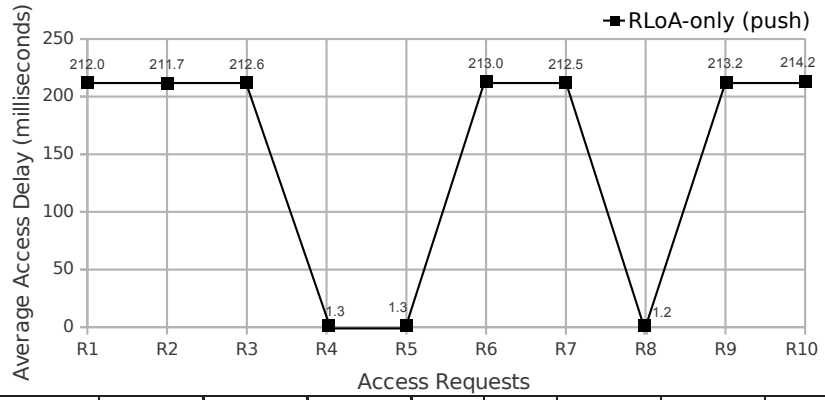
2. The PA policies operate only on *permit* rules and the algorithm used to combine these rules is *deny-override*.

3. The *push* mode is used to access the resources' OLoA policy for the 10 access requests.

Furthermore, the experiment assumes the access control decision for R4, R5 and R8 is *deny*, because the access requester does not satisfy the LoA requirements of the corresponding resource objects (i.e. $RLoA < OLoA$ for the required objects in R4, R5, and R8). On the other hand, the other access requests assume $RLoA \geq OLoA$ (i.e. *grant* decision[5]). Figure 6.11 presents the results of the experiment.

Two different sets of results can be seen from Figure 6.11. The first set is named as A, which includes R1, R2, R3, R6, R7, R9, and R10. All access requests in this set have their RLoA_{Bob} values greater than or equal to the required resource OLoA value. Set B, on the other hand, contains R4, R5 and R8, for which RLoA_{Bob} is less than the OLoA values of the required objects. As can be seen from the figure, the average AADs of set A is about 212.73 milliseconds and for set B is about 1.26 milliseconds. This means the average AAD for set B is

[5]Assuming that evaluating the PA policy produces a *grant* decision as well.



	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10
RLoA-only$_{push}$	212.0	211.7	212.6	1.3	1.3	213.0	212.5	1.2	213.2	214.2

Figure 6.11: The RLoA Value Effect on the RLoA-only$_{push}$ AAD

about 0.59% of the value for set A. This reveals that the system takes less time to *deny* than *grant*. This is because in this case, the system only needs to access the resources' OLoA policy and if the system recognises that $RLoA < OLoA$, it rejects the request. Thus, it does not need to evaluate the PA policy saving a considerable amount of processing time. This result has an interesting security implication that needs further investigation.

The previous experimental results show that the use of the RLoA-only mode makes the processing of access requests significantly efficient in case that $RLoA < OLoA$. This may indicate that the RLoA-only mode is more resilient to DoS attacks than the basic-RBAC mode. To investigate this observation further, two more experiments have been conducted. The first runs the basic-RBAC mode, while the second runs the RLoA-only mode.

The first experiment assumes Eve has successfully impersonated Bob and managed to get into the RBAC system. For example, Eve may have worked out Bob's password and logged into the system using Bob's e-ID credential. Eve then assumes Bob's role (i.e. *Staff*). At this point, Eve can launch a DoS attack on the system. For example, she may request as many resource objects as possible,

or may execute a script that sends requests at a rate of, say, 100,000 per second, to access the same resource object. For each of these requests, the system needs to check the PA policy before a *deny* or *grant* decision is made.

The experimental results show the average $AAD_{basic-RBAC}$ per access request is about 200.65 milliseconds. Thus, the basic-RBAC mode takes about 5.57 hours to process the 100,000 access requests. It is worth emphasising that the 5.57 hours is the time it takes for the system to *grant*/*deny* an access request made by Eve. During the 5.57 hour interval, the system is too busy to serve any other requests, thus a legitimate user may not be able to access the system resources. In addition, as Eve has successfully impersonated Bob's identity, Eve will be able to access the resource objects that are granted to the *Staff* role. These are two severe vulnerabilities in the basic-RBAC mode (i.e. the DoS attack and the impersonation attack).

In the second experiment, when Eve has successfully used Bob's password credential to log into the system, similar to the case in the first experiment, Eve will assume Bob's role (i.e. *Staff*). This is because, in the RLoA-only mode, CRAAC uses users' static credentials to assign initial roles. Eve can then launch the 100,000 access requests. For each of these requests, the RLoA-only mode will perform a LoA evaluation (i.e. $RLoA \geq OLoA$) before checking the actual PA policy. To succeed in this evaluation, Eve would have to compromise more access control barriers, such as passing through a location-based authentication service and/or possessing other stronger authentication credentials. Otherwise, Eve's requests will be denied. Denying an access request at this stage takes about 1.26 milliseconds. Thus, the system will take about 2.16 minutes in total to process the 100,000 access requests provided that Eve can not acquire a sufficiently higher[6] RLoA value. Therefore, the CRAAC system running the RLoA-only mode can

[6]in comparison with the requested resource object OLoA

recover sooner from DoS attacks than the basic-RBAC mode.

6.6 Chapter Summary

This chapter has proposed an access control architecture, along with its components to realise the vision of the LoA-linked access control (i.e. CRAAC). The most notable service supported by the architecture is the LoA derivation service that estimates a requester's level of assurance based upon the requester's real-time contextual information. This level of assurance value is then fed into the CRAAC authorisation decision engine, thus achieving context-risk-aware access control. Extensibility, generality, modularity, high level functional encapsulation, and transparency are among the important requirements for the design of the CRAAC architecture. To the authors' best knowledge, the CRAAC architecture is the first context-aware access control architecture designed for UbiComp environments, which has linked a subject's LoA to the sensitivity level of a requested resource object.

This chapter has also reported experiments that evaluate the RLoA-only mode against the RBAC model in terms of AADs. The experimental results reveal that a large proportion of the average access delay in the RLoA-only mode is actually caused by accessing and parsing the PA policy as well as the resources' OLoA policy. The larger the PA policy, the higher the AADs. The results also show that the additional functionality of the RLoA-only mode only contributes a relatively small level of overhead to the average access delay caused by the basic RBAC functions. Using the *push* mode to access the resources' OLoA policy can further reduce this overhead making it almost negligible. The most interesting security finding from these experiments is that the RLoA-only mode is more resilient to

both DoS[7] and impersonation attacks than the basic RBAC model.

[7]In case of denying an access request due to insufficient assurance in a subject trying to gain access to a resource object

Chapter 7

The AttributeLoA-only Mode

7.1 Chapter Introduction

As the proposed CRAAC architecture, described in Chapter 6, supports the RLoA-only mode, it should also support the other modes. This chapter describes the CRAAC architecture along with its components that support the AttributeLoA-only mode services. In addition, the chapter investigates the performance of the AttributeLoA-only mode. The investigation focuses on two fundamental factors: 1) the effect of the PA policy size, and 2)the effect of the number of contextual attributes a subject has subscribed to.

The remaining part of this chapter is structured as follows: -

- Section 7.2 shows the main differences between the AttributeLoA-only and RLoA-only modes in terms of the architectural components used.
- Section 7.3 shows how the AttributeLoA-only mode uses the CRAAC architecture to govern access control using the individual contextual attribute LoA values.
- Section 7.4 discusses the potentials and concerns of the AttributeLoA-only

mode.

- Section 7.5 investigates the performance of the AttributeLoA-only mode.
- Section 7.6 summarises the chapter.

7.2 CRAAC Architecture and the AttributeLoA-only Mode

In the AttributeLoA-only mode, access to sensitive resources is governed by the individual contextual attribute LoA values. There are differences between the AttributeLoA-only mode and the RLoA-only mode (described in Chapter 6) in terms of the resources' OLoA policy and the CRAAC architectural components used.

When the AttributeLoA-only mode is used, the resources' OLoA policy written for the RLoA-only mode is no longer applicable. In the RLoA-only mode, each resource object is tagged with a single OLoA value that a subject has to satisfy to gain access to the resource object. However, in some application scenarios, a resource provider may specify a particular LoA requirement for each contextual attribute. As shown in Figure 5.1, the access requirement for a resource object (i.e. Printer) can be specified in terms of the LoA values associated to one or more individual contextual attributes (e.g. AH, CS, etc). This is the AttributeLoA-only mode of working.

As seen in Figure 7.1, the main difference, in the architectural components used, between the AttributeLoA-only and RLoA-only modes are: 1) the use of the LoA-aware PDPs, 2) the use of $\text{PDP}_{AttributeLoA}$, and 3) the removal of the LoA Aggregator, PDP_{RLoA-I}, and $PDP_{RLoA-II}$.

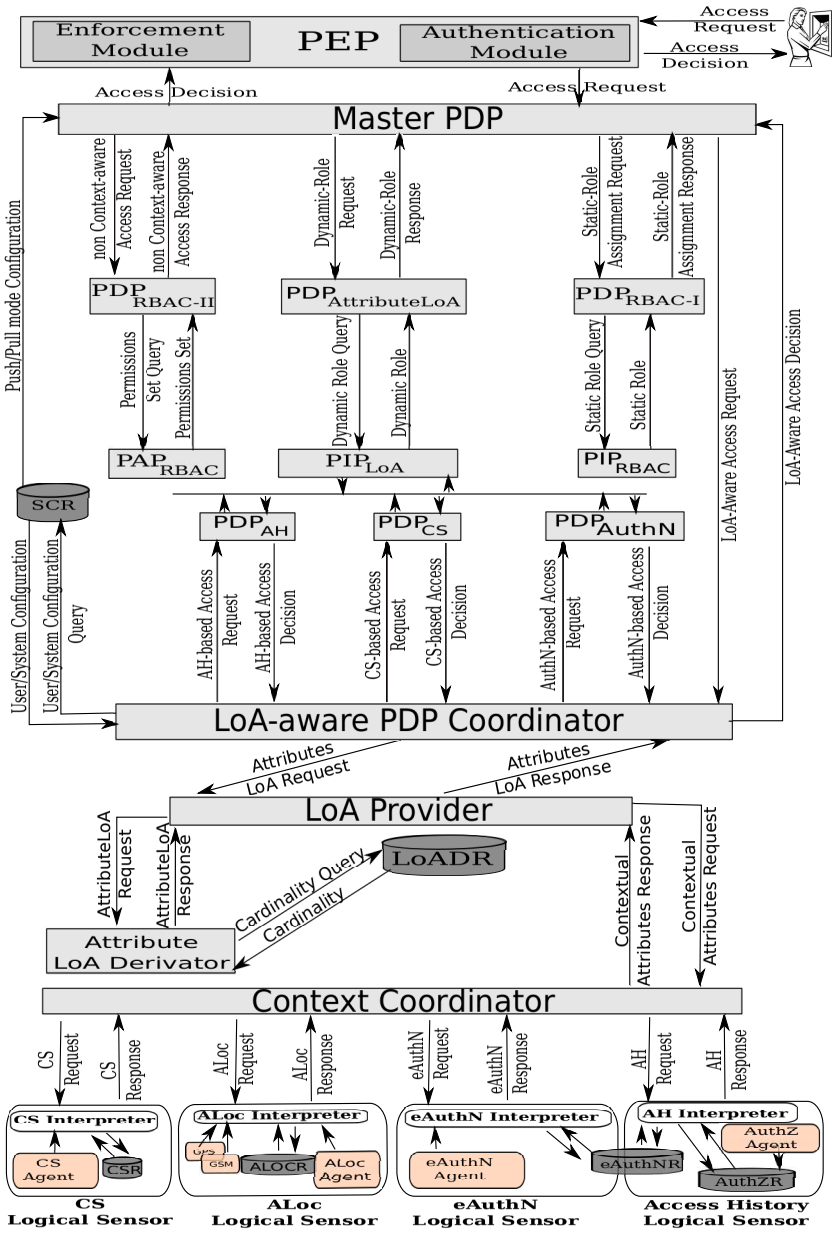


Figure 7.1: The CRAAC Architecture: the AttributeLoA-only Mode

Since resource access in the AttributeLoA-only mode is controlled by the individual attributeLoA values, the $\text{PDP}_{AttributeLoA}$ is proposed to replace PDP_{RLoA-I} in order to generate a subject's dynamic role based on the subject's individual contextual attribute LoA values instead of RLoA as in PDP_{RLoA-I}. The LoA Aggregator is disabled in the AttributeLoA-only mode as there is no need for

aggregating the individual attribute LoA values into an RLoA value as in the RLoA-only mode. Thus, the main function of the LoADI, in the AttributeLoA-only mode, is to receive the contextual attribute values from the CMI, derives relevant LoA values, and sends the LoA values back to the LoA-aware PDP Coordinator. The LoA-aware PDP Coordinator will then submit the LoA values to the corresponding LoA-aware PDP (i.e. PDP_{CS}, PDP_{AuthN} or PDP_{AH}). A LoA-aware PDP will compare the received LoA value of the subject against that of the resource object in order to reach a partial access control decision that will be sent to the LoA-aware PDP Coordinator. The LoA-aware PDP Coordinator aggregates all the partial access control decisions made by the multiple LoA-aware PDPs into one LoA-aware access control decision. Actually, the set of LoA-aware PDPs, in this mode, replaces $\text{PDP}_{RLoA-II}$ in order to generate access control decisions based upon the individual attributeLoA values instead of the RLoA values.

7.3 AttributeLoA-only Mode Data-Flow

The main difference in the work-flow between the AttributeLoA-only mode and the RLoA-only mode is that the former does not address RLoA aggregation. Rather, it uses the subordinate LoA-aware PDPs in order to generate partial access control decisions based on the LoA values of the contextual attributes used. For example, to take an access control decision based on the authN contextual attribute, the PDP_{authN} evaluates the OLoA_{authN} required by a resource object against the LoA_{authN} of an access requester. The AttributeLoA-only mode provides a new process to aggregate all subordinate LoA-aware PDPs partial access control decisions into one final LoA-aware access control decision. This process is performed by the LoA-aware PDP Coordinator. In fact, this process has not been

seen in the RLoA-only mode[1], since the RLoA-only mode does not take decision in terms of the individual attribute LoA values. This is depicted in Figure 7.2.

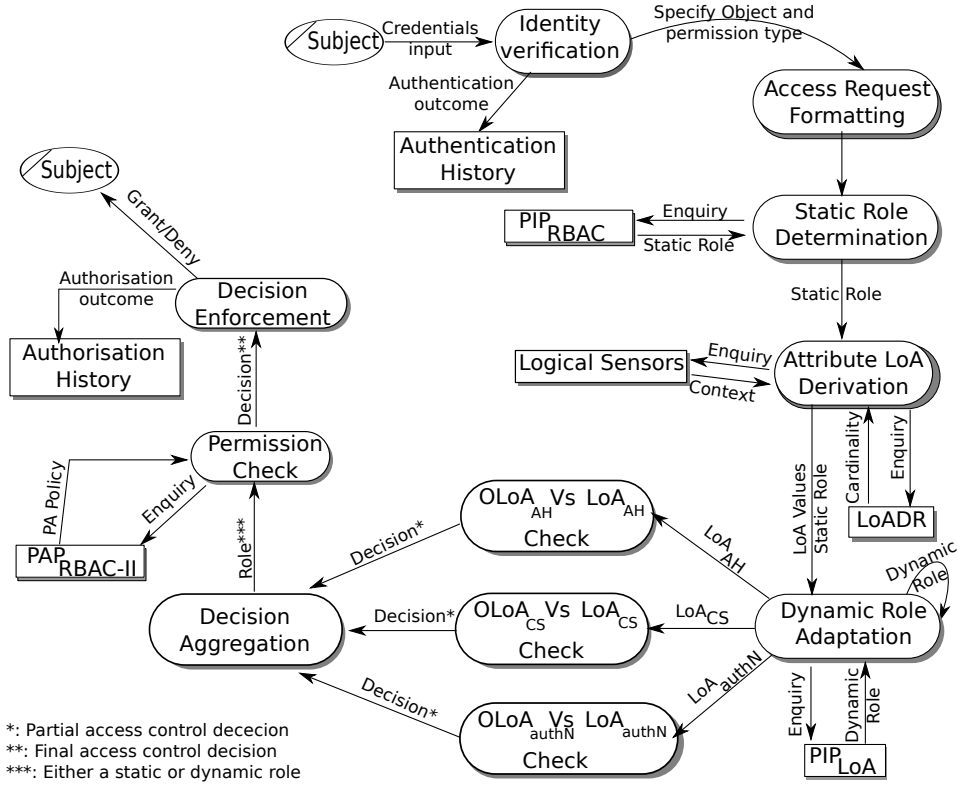


Figure 7.2: Data-flow in the AttributeLoA-only Mode

7.4 Potentials and Concerns

The AttributeLoA-only mode provides new services in comparison with the RLoA-only mode. Those include:

1. The ability to assign different contextual attribute sets to different services or resource objects.

   This allows a service provider to require a subject to subscribe to a particular subset of a contextual attribute set and use the corresponding LoA values to control access to a resource object.

[1]See Figure 6.3 for more detail

2. To allow a service provider to specify a certain LoA requirement on a particular contextual attribute set in order to access its resource objects. In other words, the contextual attributes may have varying levels of impact on the overall access control decision. For example, a service provider may value the location contextual attribute over the channel security attribute when releasing its resource object(s).

3. The AttributeLoA-only mode maps naturally to the existing context-aware access control solutions largely discussed in Chapter 3. For example, if only a location-aware PDP is used, the AttributeLoA-only mode will correspond to the SRBAC model [39]. This is accomplished of course after the location information is converted to the corresponding LoA.

Two issues need to be discussed in the AttributeLoA-only mode: context-dependency and potential communication overhead. Since an access control policy is, conceptually, expressed in terms of the individual contextual attribute types and their corresponding LoA values, adding a new or remove an obsolete contextual attribute requires alteration in the architectural components. For example, adding a temporal attribute requires the addition of a new LoA-aware PDP (i.e. $PDP_{temporal}$) as well as modifying the LoA-aware PDP Coordinator to acknowledge the new PDP. In fact, this drawback is inherited from the traditional context-aware access control models[2]. However, the level of context-dependency in the AttributeLoA-only mode differs from that of the traditional context-aware access control models. The context-dependency in the former isn't expected to hinder extensibility. This is because CRAAC uses a standard interface between the LoA-aware PDP Coordinator and the set of subordinate LoA-aware PDPs. Currently, CRAAC uses XACML 2.0 as the standard interface. Thus, adding a

[2]See Section 3.4 for more detail

new LoA-aware PDP should only exhibit a minimum modification in the other architectural components. Also, the AttributeLoA-only mode hides the contextual data representation from the authorisation engine, thus providing a higher level of encapsulation than that of the CAAC-based solutions.

The communication overhead in the AttributeLoA-only mode is expected to be generally higher than that of the RLoA-only mode. This is because, in the former mode of working, for every contextual attribute used by the system, a separate LoA-aware PDP is required. This is in contrast to the RLoA-only mode, which requires only a single LoA-linked authorisation decision point (i.e. $\text{PDP}_{RLoA-II}$). Thus, the more attributes used, the more PDPs are required, hence the higher the communication overhead. The level of overhead versus the number of LoA-aware PDPs will be experimentally assessed.

One may argue that the overall policy complexity of the model may be significant on the model performance. If CRAAC policy is compared against that of the CAAC-based model, CRAAC policy is simpler and readable. Only one (i.e. in RLoA-only mode) constraint is expressed in the policy. In addition, CRAAC can easily use any policy language without a significant modification as the LoA constraint is a simple data type (i.e. double) that most policy languages support. This is a major contrast to the existing CRAAC-based solutions.

7.5 AttributeLoA-only Mode Performance Evaluation

This section reports the experimental investigations of the AttributeLoA-only mode performance. The main objective of the experiments is to answer the question: What is the additional cost (i.e. in terms of AADs) a subject has

to endure in the AttributeLoA-only mode in comparison with the RLoA-only mode? Two factors are expected to affect the performance of the AttributeLoA-only mode: the PA policy size and the number of contextual attributes used (i.e. the number of LoA-aware PDPs required).

7.5.1 The Effect of the PA Policy Size

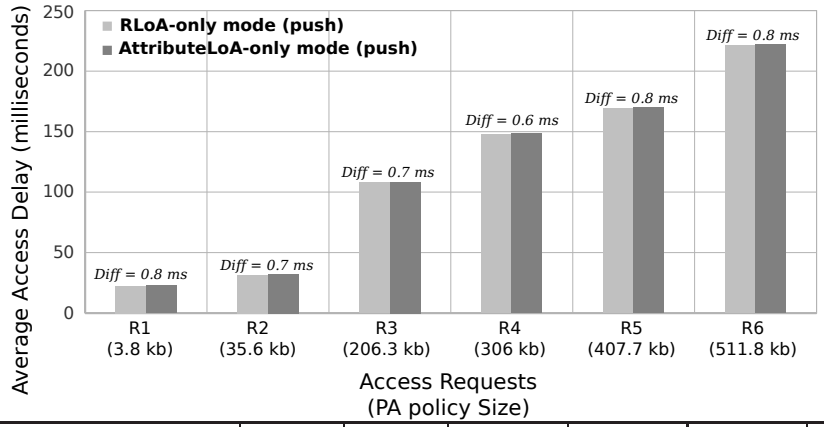
This experiment investigates the effect of the PA policy size on the AADs in the AttributeLoA-only mode and compares the results against those from the RLoA-only mode. The following configurations have been used in this experiment:

1. The *push* mode is used to access the resources' OLoA policy.
   As the focus of this experiment is to investigate the effect of the PA policy size on AADs, the use of the *push* mode eliminates the effect of the resources' OLoA policy size.

2. In the experiment, a subject, Bob, with one enabled role, Staff, makes six access requests to six different resource objects. The six access requests are denoted as R1 through R6 with PA policy sizes of 3.8, 35.6, 206.3, 306.0, 407.7, and 511.8 KBytes, respectively.

3. Only *permit* rules are used in policies and the algorithm used to combine the rules is *deny-override*. The implication of this assumption is that every rule contained in the PA policy will be evaluated before making a decision. The experiment also assumes Bob's contextual attributes LoA values are always greater than the required LoA values of the requested resource object across the six access requests. This assumption ensures the execution of CRAAC always reaches the stage where the PA policy files are opened and all policy rules contained in the files are evaluated. In this way, the experiment can investigate the precise effects of various PA policy sizes on the AADs.

Figure 7.3 shows two sets of AADs measured in milliseconds against the six different access requests. One set is from the RLoA-only mode and the other is from the AttributeLoA-only mode. The following observation can be made from the figure.



	R1	R2	R3	R4	R5	R6
RLoA-only$_{push}$	22.35	31.67	107.72	147.77	168.93	221.19
AttributeLoA-only$_{push}$	22.67	31.92	108.37	148.36	169.73	221.95

Figure 7.3: AttributeLoA-only Vs RLoA-only: the Effect of the PA Policy Size

In both AttributeLoA-only and RLoA-only modes, the AADs increase steadily as the PA policy size increases. The differences between the two sets of AADs are negligible (i.e. about 0.48% averaged across the six access requests). This result demonstrates that the major AAD contributions are from the components or services that are common in both modes of working. The additional services/components used in the AttributeLoA-only mode contribute fractionally to the overall AADs. In fact, the amount of overheads contributed by the AttributeLoA-only mode is consistent and almost the same across different PA policy sizes. The overhead appears to be independent of the PA policy size. This observation reveals that the AttributeLoA-only mode performance is almost the same as that of the RLoA-only mode; this was not anticipated. After an investigation we have found that the reason for this is the use of *push* mode in retrieving

the PA policies. Pushing the resources' OLoA policy in this mode makes all the LoA-aware PDPs operate on memory not on disk. In other words, a LoA-aware PDP needs just a memory search to get the required resource object's LoA requirement (i.e. OLoA) in order to generate a partial LoA-aware access control decision. However, if the LoA-aware PDP uses the *pull* mode to access a resources' OLoA policy file, the additional overhead incurred by pulling the policy may be significant. To verify this hypothesis, an experiment has been performed, in which the LoA-aware PDPs pull the corresponding policy. In this experiment, the average AAD across the 6 access requests increases by about 23% than that of the RLoA-only mode. This increase is significant and complies with our expectation. As a conclusion, the pull mode is not recommended to be used when the CRAAC model is configured in the AttributeLoA-only mode.

7.5.2 The Effect of the Number of Contextual Attributes

The main objective of this experiment is to investigate the effect of the number of contextual attributes, a subject has subscribed to, on the AADs in the AttributeLoA-only mode. The experiment evaluates the effects of 6 contextual attribute set sizes (i.e. 1, 2, 4, 8, 16, and 32) on the AADs in the AttributeLoA-only mode. This, in fact, evaluates the effect of the number of LoA-aware PDPs on the overall AADs in the AttributeLoA-only mode. The experiment uses the following configuration:

1. The same resource object is accessed by the same access requester (i.e. Bob) in the 6 cases of contextual attribute sizes. Thus, the effect of the PA policy size on the AADs should be the same across the 6 cases. In addition, Bob's contextual attributes LoA values are always greater than the required LoA values of the requested resource object across the six access requests.

2. Only *permit* rules are used in the PA policies and the algorithm used to combine the rules is *deny-override*. The implication of this assumption is that every rule contained in the PA policy file will be evaluated before making an access control decision.

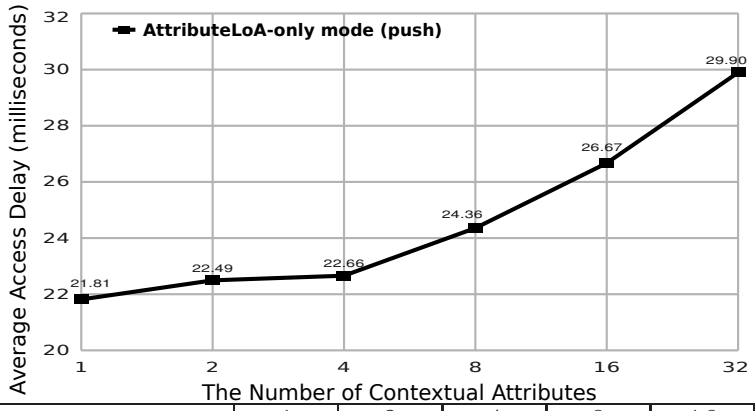
3. The resources' OLoA policy is accessed using the *push* mode.

4. All contextual attributes in Bob's contextual attributes set are of the same cardinality and rank value. This ensures the effect of the *ROCs* (i.e. used for L2WC) is almost the same, thus eliminating the effect of the types of the contextual attributes used and focuses only on their number.

Figure 7.4 shows the results of the experiment. It shows a set of AADs measured in milliseconds against 6 different contextual attribute set sizes. It can be seen from the figure that the more contextual attributes used, the longer the delay an access requester would have to endure. For example, when the contextual attribute set size is 1, the AAD is about 21.81 milliseconds and when it is 2, the AAD is about 22.50 milliseconds. This is about 3.13% increase in AAD, which may not be significant. However, when the size increases from 1 to 8, the increase in AAD is about 11.66%, and when the size increases from 1 to 32 the AAD increase is about 37%. This should not be considered negligible.

7.6 Chapter Summary

This chapter has shown the CRAAC architecture in the AttributeLoA-only mode. The differences between the architectural components used in the RLoA-only mode and those used in the AttributeLoA-only mode are: 1) the use of the LoA-aware PDPs, 2) the use of $\text{PDP}_{AttributeLoA}$, and 3) the removal of the LoA Aggregator, PDP_{RLoA-I}, and $PDP_{RLoA-II}$. The most notable potential of the



	1	2	4	8	16	32
AttributeLoA-only$_{push}$	21.81	22.50	22.66	24.36	26.67	29.90

Figure 7.4: AttributeLoA-only: the Effect of the Contextual Attribute Set Size

AttributeLoA-only mode is that it maps naturally to the existing context-aware access control solutions discussed in Chapter 3. This chapter has also reported experiments that evaluate the AttributeLoA-only mode in terms of AADs.

The experimental results reveals that the AttributeLoA-only mode performance is comparable to the RLoA-only mode using the *push* policy retrieval mode. When the *pull* mode is used, the additional overhead considerably shoots high. Thus, the *pull* mode is not recommended when CRAAC is configured in the AttributeLoA-only mode. The number of contextual attributes a subject has subscribed to is another important factor that affect the AttributeLoA-only mode AADs. The experimental investigation shows that the more contextual attributes, the higher AAD a subject will endure.

Chapter 8

Conclusion and Future Work

This thesis addressed challenges in access control for UbiComp environments. These challenges include:

- How to capture an access requester's dynamic contextual information and feed it into the authorisation decision engine seamlessly at run-time.
- How to accommodate new contextual attributes without imposing considerable modifications on the underlying access control service.
- How to support fine-grained access control of resource objects with varying sensitivity levels in a context-aware environment such as UbiComp.
- How to deal with contextual information that is expressed using different representation models.
- How to minimise the effect of compromising the access control policy store.

8.1 Conclusion

The Context-Risk-Aware Access Control model (CRAAC) for UbiComp environments is the novel contribution of this thesis. CRAAC is a LoA-linked access

control model that controls access to resources with varying levels of sensitivity based, indirectly, upon an access requester's contextual information. The contextual information is used to compute the access requester's LoA value that is used as the only constraint to control access to the protected resources. CRAAC supports fined-grained access control, since it, virtually, accommodates any set of contextual attributes. CRAAC provides flexibility in adding new contextual attributes or removing obsolete ones. This flexibility will not considerably affect the underlying access control system. In other words, the level of modification in the underlying access control system is not significant. CRAAC supports four modes of working to accommodate different access control requirements. The derivation of an access requester's aggregate LoA value (i.e. RLoA) is controlled by the relationships amongst the access requester's set of contextual attributes (i.e. *Elevating* or *Weakest-Link*).

The CRAAC safety is expected to be higher than that of the RBAC model, since CRAAC utilises additional constraints (i.e. LoA constraints). It is also evident that CRAAC is more resilient to DoS attacks than the traditional RBAC model in denying access to a resource object when the attacker can not acquire a sufficiently high RLoA value. The thesis has reported experiments that evaluate the RLoA-only, basic-RBAC, and AttributeLoA-only modes in terms of AADs. The experimental results reveal that a large proportion of the average access delay in the RLoA-only mode is actually caused by accessing and parsing the PA policy as well as the resources' OLoA policy. And so for the AttributeLoA-only mode. The larger the PA policy file, the higher the AADs. The results also shows that the additional functionality of the RLoA-only mode only contributes a small level of overhead to the average access delay caused by the basic RBAC functions. Using the *push* mode to access the resources' OLoA policy can further reduce the overhead making it almost negligible. Moreover, the experimental

results show that the AttributeLoA-only mode performance is comparable to the RLoA-only mode using the *push* policy retrieval mode. When the *pull* mode is used, the additional overhead is considerably high. Thus, the *pull* mode is not recommended when CRAAC is configured in the AttributeLoA-only mode.

The following lists the novel contributions of the research presented in this thesis: -

1. The proposal of the CRAAC model.

   CRAAC is an adaptive LoA-linked access control model. It controls access to resources with varying levels of sensitivity based upon the state of an access requester's contextual information. It supports adaptive access control decisions, since an access control decision for the same access requester on the same resource object may vary each time. The variation depends on the level of assurance of the access requester, which is based upon the access requester's current contextual information.

   - CRAAC supports fined-grained access control, since it, virtually, accommodates any set of contextual attributes. This level of abstraction is achieved by the use of a trust-related parameter (i.e. Requester's Level of Assurance (RLoA)). In addition, in controlling access to a resource object, CRAAC accommodates the resource object's sensitivity level in order to provide a more fine-grained access control.
   - CRAAC is flexible; adding new contextual attributes or removing obsolete ones will not significantly affect the underlying access control system.
   - CRAAC supports four modes of working to accommodate different access control requirements.

2. LoA calculations

    - The identification of the contextual attributes that may have an impact on a subject's LoA.
    - The LoA quantification of the corresponding contextual attributes. This is performed mimicking the NIST LoA_{eToekn} work in order to convert the contextual attribute values into LoA ranks.
    - Surveying the possible LoA to weight conversion methods and adopting ROC for the conversion (i.e. L2WC).
    - Proposing two methods for the RLoA aggregation (i.e. *Weakest-Link* and the *Elevating*).

3. An access control architecture along with its components to support the novel CRAAC services. The architecture supports the four modes of working.

4. CRAAC Evaluation

    - Prototype-based performance evaluation of the CRAAC model.
    - Investigating the safety of the CRAAC model in terms of the constraints used.
    - Investigating the denial of service and impersonation attacks.

8.2 Future Work

Through research, we have identified the following issues that require further investigation:

- How to quantify and derive the assurance level of an access requester's authorisation and authentication history. In other words, how could we quantitatively assess the authorisation and authentication transactions of an access requester for a past period? Currently, CRAAC assumes this is done at the corresponding access history logical sensor.

- CRAAC management in a distributed setting.

  The CRAAC architecture has been evaluated in a centralised setting by which all the CRAAC architectural components are placed in one site (i.e. one PC). The performance of the CRAAC model when its architecture is distributed should be evaluated. This also may encompass the possibility of using multiple authorisation servers. The proposed architecture of the CRAAC model may not be readily adopted in a distributed setting, thus may add more overhead/complexity. In addition, using a distributed setting may impose more security vulnerabilities and loopholes an attacker could utilise. For example, how to trust a piece of contextual information that is provided by a remote context provider located in another domain? Such a concern needs to be investigated before using CRAAC in a distributed setting.

- Policy Languages.

  For interoperability with the well-known existing authorisation solutions, there is a need to implement more standard PDP interfaces (e.g. PERMIS).

- Generic Authorisation Architecture

  Can CRAAC fit in any application domain even those non context-aware? The possibility of the current CRAAC architecture to be adapted, to cope with diversified access control requirements, needs to be assessed. For example, can CRAAC be used to address the GRID access control requirements?

What are the level of modifications in the current CRAAC architecture required for this? In other words, how to extend the CRAAC functionality without compromising its features. For example, a basic requirement is to have an obligation service. The obligation service is required to perform temporal obligations (i.e. before, with, and after). A generic implementation of the obligation service is challenging as obligations are application-dependant.

- XACML LoA-aware profile. This is a challenging task as creating an XACML profile for LoA-aware access control policies not only requires a modification in the current XACML standard but also providing a code library that evaluates such policies.

- How to calculate the trust level of both the contextual providers and their provided contextual information. In other words, the contextual data management will not be assumed as a part of the TCB.

- Investigating more vulnerability that may CRAAC may have. This may include convert channel attacks, distributed DoS attack, etc.

Bibliography

[1] W. Burr, D. Dodson, and W. T. Polk, "Special publication 800-63: Electronic authentication guideline v1.0.2," National Institute for Standards and Technology, Tech. Rep., April 2006.

[2] A. Ahmed and N. Zhang, "A context-risk-aware access control model for ubiquitous environments," in *IMCSIT' 2008: International Multiconference on Computer Science and Information Technology.*, October 2008, pp. 775–782.

[3] T. Saaty, *The Analytic Hierarchy Process: Planning, Priority Setting, Resource Allocation.* McGraw-Hill, New York, 1980.

[4] V. C. Hu, F. David, and K. D. Richard, *Assessment of access control systems [electronic resource] / Vincent C. Hu, David F. Ferraiolo, D. Rick Kuhn.* U.S. Dept. of Commerce, National Institute of Standards and Technology, [Gaithersburg, Md.] :, 2006.

[5] A. Corradi, R. Montanari, and D. Tibaldi, "Context-based access control management in ubiquitous environments," *Network Computing and Applications, IEEE International Symposium on*, vol. 0, pp. 253–260, 2004.

[6] M. Anisetti, C. Ardagna, V. Bellandi, E. Damiani, S. D. C. di Vimercati, and P. Samarati, "Openambient: a pervasive access control architecture," in

Proc. of ETRICS Workshop on Security in Autonomous Systems (SecAS), vol. Friburg, Germany, June 2006.

[7] M. Román, C. Hess, R. Cerqueira, A. Ranganathan, R. H. Campbell, and K. Nahrstedt, "Gaia: a middleware platform for active spaces," *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 6, no. 4, pp. 65–67, October 2002.

[8] J. Chin, N. Zhang, A. Nenadic, and O. Bamasak, "A context-constrained authorisation (cocoa) framework for pervasive grid computing," *The Wireless Networks (WINET)*, September 2008.

[9] J. B. Filho and H. Martin, "A generalized context-based access control model for pervasive environments," in *SPRINGL '09: Proceedings of the 2nd SIGSPATIAL ACM GIS 2009 International Workshop on Security and Privacy in GIS and LBS*. New York, NY, USA: ACM, 2009, pp. 12–21.

[10] E. Chen, Y. Shi, D. Zhang, and G. Xu, "A programming framework for service association in ubiquitous computing environments," in *Information, Communications and Signal Processing, 2003 and the Fourth Pacific Rim Conference on Multimedia. Proceedings of the 2003 Joint Conference of the Fourth International Conference on*, vol. 1, December 2003, pp. 202–207.

[11] R. J. Hulsebosch, A. Salden, M. Bargh, E. Peter, and J. Reitsma, "Context sensitive access control," in *SACMAT '05: Proceedings of the tenth ACM Symposium on Access Control Models and Technologies*. New York, NY, USA: ACM Press, 2005, pp. 111–119.

[12] S. Intille, "The goal: Smart people, not smart homes," in *Proceedings of the International Conference on Smart Homes and Health Telematics*, 2006.

[13] G. D. Abowd, "Software engineering issues for ubiquitous computing," in *ICSE '99: Proceedings of the 21st international conference on Software engineering*. Los Alamitos, CA, USA: IEEE Computer Society Press, 1999, pp. 75–84.

[14] B. W. Lampson, "Protection," *SIGOPS Oper. Syst. Rev.*, vol. 8, no. 1, pp. 18–24, 1974.

[15] D. Tor, "Rule based database access control–a practical approach," in *RBAC '97: Proceedings of the second ACM workshop on Role-based access control*. New York, NY, USA: ACM, 1997, pp. 143–151.

[16] D. F. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramouli, "Proposed nist standard for role-based access control," *ACM Transactions on Information and System Security*, vol. 4, no. 3, p. 224–274, August 2001.

[17] M. J. Moyer and M. Ahamad, "Generalized role-based access control," in *ICDCS '01: Proceedings of the The 21st International Conference on Distributed Computing Systems*. Washington, DC, USA: IEEE Computer Society, April 2001, pp. 391–398.

[18] S.-C. Chou, "An rbac-based access control model for object-oriented systems offering dynamic aspect features," *IEICE - Trans. Inf. Syst.*, vol. 88, no. 9, pp. 2143–2147, 2005.

[19] S.-H. Park, Y.-J. Han, and T.-M. Chung, "Context-role based access control for context-aware application," in *High Performance Computing and Communications*, vol. 4208. Springer Berlin / Heidelberg, September 2006, pp. 572–580.

[20] A. Dey, "Understanding and using context," *Personal Ubiquitous Comput.*, vol. 5, no. 1, pp. 4–7, 2001.

[21] S. Lachmund, T. Walter, L. Gomez, L. Bussard, and E. Olk, "Context-aware access control; making access control decisions based on context information," in *Mobile and Ubiquitous Systems: Networking & Services, 2006 Third Annual International Conference on*, July 2006, pp. 1–8.

[22] A. Manzoor, H.-L. Truong, and S. Dustdar, "On the evaluation of quality of context," in *Smart Sensing and Context*, ser. Lecture Notes in Computer Science, D. Roggen, C. Lombriser, G. Tröster, G. Kortuem, and P. Havinga, Eds. Springer Berlin / Heidelberg, 2008, vol. 5279, pp. 140–153, 10.1007/978-3-540-88793-5_11. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-88793-5_11

[23] R. Sandhu and P. Samarati, "Access control: Principles and practice," *IEEE Communications Magazine*, vol. 32, no. 9, pp. 40–48, 1994. [Online]. Available: http://www.list.gmu.edu/journals/commun/i94ac%28org%29.pdf

[24] I. Foster, C. Kesselman, G. Tsudik, and S. Tuecke, "A security architecture for computational grids," in *CCS '98: Proceedings of the 5th ACM conference on Computer and communications security.* New York, NY, USA: ACM, 1998, pp. 83–92.

[25] D. of Defense, "Department of defense trusted computer system evaluation criteria," Department of Defense Standard, Tech. Rep., December 1985. [Online]. Available: http://csrc.nist.gov/publications/history/dod85.pdf

[26] S.-p. Li, S.-z. Wu, and T. Guo, "The consistency of an access control list," in *ICICS '02: Proceedings of the 4th International Conference on Information*

and Communications Security. London, UK: Springer-Verlag, 2002, pp. 367–373.

[27] B. W. Lampson, "A note on the confinement problem," *Commun. ACM*, vol. 16, no. 10, pp. 613–615, 1973.

[28] S. J. Murdoch, "Covert channel vulnerabilities in anonymity systems," University of Cambridge, Computer Laboratory, Technical Report 706, 2007. [Online]. Available: http://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-706.pdf

[29] V. Gligor, S. Gavrila, and D. Ferraiolo, "On the formal definition of separation-of-duty policies and their composition," in *Proceedings of the IEEE Symposium on Security and Privacy*, Berkeley, CA, may 1998, pp. 172–183.

[30] D. Ferraiolo and R. Kuhn, "Role-based access controls," National Institute of Standards and Technology, Technology Administration, U.S. Department of Commerce,Gaithersburg, Md. 20899 USA, Tech. Rep., January 1995. [Online]. Available: http://hissa.nist.gov/rbac/paper/rbac1.html

[31] R. S. Sandhu, "Future directions in role-based access control models," in *MMM-ACNS '01: Proceedings of the International Workshop on Information Assurance in Computer Networks*. London, UK: Springer-Verlag, 2001, pp. 22–26.

[32] S. Osborn, R. Sandhu, and Q. Munawer, "Configuring role-based access control to enforce mandatory and discretionary access control policies," *ACM Transactions on Information and System Security (TISSEC)*, vol. 3, no. 2, pp. 85–106, May 2000.

[33] S. Oh and S. Park, "Task-role-based access control model," *Information Systems*, vol. 28, no. 6, pp. 533 – 562, 2003. [Online]. Available: http://www.sciencedirect.com/science/article/B6V0G-46WW2TN-1/2/33bb5b5ba058905dddb35ae00e4cabea

[34] J. gang Chen, R. chuan Wang, and H. yan Wang, "The extended rbac model based on grid computings," *The Journal of China Universities of Posts and Telecommunications*, vol. 13, no. 3, pp. 93 – 97, 2006. [Online]. Available: http://www.sciencedirect.com/science/article/B8H14-4NHV7N7-N/2/ee8e0849c1906b737b767500c7f01ccc

[35] M. J. Covington, M. J. Moyer, and M. Ahamad, "Generalized role-based access control for securing future applications," in *In Proceedings of the 23rd National Information Systems Security Conference (NISSC)*, Baltimore, Maryland, USA, October 2000.

[36] M. J. Covington, P. Fogla, Z. Zhan, and M. Ahamad, "A context-aware security architecture for emerging applications," in *ACSAC '02: Proceedings of the 18th Annual Computer Security Applications Conference*. Washington, DC, USA: IEEE Computer Society, 2002, p. 249.

[37] E. Bertino, P. A. Bonatti, and E. Ferrari, "Trbac: A temporal role-based access control model," *ACM Trans. Inf. Syst. Secur.*, vol. 4, no. 3, pp. 191–233, 2001.

[38] J. Joshi, E. Bertino, and A. Ghafoor, "Hybrid role hierarchy for generalized temporal role based access control model," in *COMPSAC '02: Proceedings of the 26th International Computer Software and Applications Conference on Prolonging Software Life: Development and Redevelopment*. Washington, DC, USA: IEEE Computer Society, 2002, pp. 951–956.

[39] F. Hansen and V. Oleshchuk, "Srbac: a spatial role-based access-control model for mobile systems," in *In Proceedings of the 7th Nordic Workshop on Secure IT Systems (NORDSEC'03). Gj'vik, Norway*, 2003, pp. 129–141.

[40] H. Zhang, Y. He, and Z. Shi, "Spatial context in role-based access control," in *Information Security and Cryptology – ICISC 2006*, ser. Lecture Notes in Computer Science, vol. 4296, ICISC. Springer Berlin / Heidelberg, November 2006, pp. 166–178.

[41] H. Yu and E.-P. Lim, "Ltam: A location-temporal authorization model," in *Secure Data Management*, S. 2004, Ed., vol. 3178, Center for Advanced Information Systems, Nanyang Technological University, SINGAPOUR. Springer-Verlag Berlin Heidelberg 2004, 2004, p. 172–186.

[42] Y.-G. Kim, C.-J. Mon, D. Jeong, J.-O. Lee, C.-Y. Song, and D.-K. Baik, "Context-aware access control mechanism for ubiquitous applications," in *Advances in Web Intelligence*, vol. 3528. Springer Berlin / Heidelberg, May 2005, pp. 236–242.

[43] I. Ray and L. Yu, "Short paper: Towards a location-aware role-based access control model," in *SECURECOMM '05: Proceedings of the First International Conference on Security and Privacy for Emerging Areas in Communications Networks (SECURECOMM'05)*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 234–236.

[44] S. M. Chandran and J. B. D. Joshi, "Lot-rbac: A location and time-based rbac model." in *WISE*, ser. Lecture Notes in Computer Science, vol. 3806. Springer, 2005, pp. 361–375.

[45] C. A. Ardagna, M. Cremonini, E. Damiani, S. D. C. di Vimercati, and

P. Samarati, "Supporting location-based conditions in access control policies," in *ASIACCS '06: Proceedings of the 2006 ACM Symposium on Information, computer and communications security.* New York, NY, USA: ACM, 2006, pp. 212–222.

[46] S. hwa Chae, W. Kim, and D. kyoo Kim, "Role-based access control model for ubiquitous computing environment," in *Information Security Applications*, W. 2005, Ed., vol. 3786. Springer Berlin / Heidelberg, February 2006, pp. 354–363.

[47] M. L. Damiani, E. Bertino, B. Catania, and P. Perlasca, "Geo-rbac: A spatially aware rbac," *ACM Trans. Inf. Syst. Secur.*, vol. 10, no. 1, p. 2, 2007.

[48] L. Chen and J. Crampton, "On spatio-temporal constraints and inheritance in role-based access control," in *ASIACCS '08: Proceedings of the 2008 ACM symposium on Information, computer and communications security.* New York, NY, USA: ACM, 2008, pp. 205–216.

[49] M. Toahchoodee, I. Ray, K. Anastasakis, G. Georg, and B. Bordbar, "Ensuring spatio-temporal access control for real-world applications," in *SACMAT '09: Proceedings of the 14th ACM symposium on Access control models and technologies.* New York, NY, USA: ACM, 2009, pp. 13–22.

[50] M. Kirkpatrick and E. Bertino, "Context-dependent authentication and access control," in *iNetSec 2009 – Open Research Problems in Network Security*, ser. IFIP Advances in Information and Communication Technology, vol. 309. Springer Boston, November 2009, pp. 63–75.

[51] G. Zhang and M. Parashar, "Dynamic context-aware access control for grid applications," in *GRID '03: Proceedings of the 4th International Workshop*

on Grid Computing. Washington, DC, USA: IEEE Computer Society, 2003, p. 101.

[52] P. Bellavista, A. Corradi, R. Montanari, and C. Stefanelli, "Context-aware middleware for resource management in the wireless internet," *IEEE Transactions on Software Engineering*, vol. 29, no. 12, pp. 1086–1099, 2003.

[53] S. Daniel, D. Anind, and A. Gregory, "The context toolkit: aiding the development of context-enabled applications," in *CHI '99: Proceedings of the SIGCHI conference on Human factors in computing systems*. New York, NY, USA: ACM, 1999, pp. 434–441.

[54] G. Sampemane, P. Naldurg, and R. H. Campbell, "Access control for active spaces," in *ACSAC '02: Proceedings of the 18th Annual Computer Security Applications Conference*. Washington, DC, USA: IEEE Computer Society, 2002, p. 343.

[55] G. T. (GT), "http://www.globus.org/toolkit/," Web, last accessed January 2010.

[56] J. E. Bardram, "From desktop task management to ubiquitous activity-based computing," in *Integrated Digital Work Environments: Beyond the Desktop Metaphor*, V. Kaptelinin and M. Czerwinski, Eds. MIT Press, 2007, pp. 49–78. [Online]. Available: http://mitpress.mit.edu/catalog/item/default.asp?ttype=2&tid=11154

[57] Z. Zhu and R. Xu, "A context-aware access control model for pervasive computing in enterprise environments," oct. 2008, pp. 1 –6.

[58] Y. Lu, L. Zhang, and J. Sun, "Task-activity based access control for process collaboration environments," *Computers in Industry*, vol. 60, no. 6, pp. 403

– 415, 2009, collaborative Engineering: from Concurrent Engineering to Enterprise Collaboration. [Online]. Available: http://www.sciencedirect.com/science/article/B6V2D-4VY166F-1/2/845865a20271116b3aa74a141e20d243

[59] S. Zheng, D. Jiang, and Q. Liu, "A role and activity based access control model for university identity and access management system," in *Information Assurance and Security, 2009. IAS '09. Fifth International Conference*, vol. 2, aug. 2009, pp. 487 –490.

[60] L. X. Hung, S. R, J. H., R. S., Y. Weiwei, N. T. Canh, T. P., S. Lee, H. Lee, Y. Son, and F. M., "Activity-oriented access control for ubiquitous environments," January 2009, pp. 1–5.

[61] P. Argyroudis and D. O'Mahony, "ÆTHER: an Authorization Management Architecture for Ubiquitous Computing," in *Proceedings of 1st European PKI Workshop: Research and Applications (EuroPKI'04)*, ser. Lecture Notes in Computer Science, no. 3093. Samos island, Greece: Springer-Verlag, June 2004, pp. 246–259.

[62] N. E. Dimmock, "Using trust and risk for access control in global computing," University of Cambridge, Computer Laboratory, 15 JJ Thomson Avenue, Cambridge CB3 0FD, United Kingdom, PhD Thesis 643, August 2005. [Online]. Available: http://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-643.pdf

[63] N. N. Diep, L. X. Hung, Y. Zhung, S. Lee, Y.-K. Lee, and H. Lee, "Enforcing access control using risk assessment," in *ECUMN '07: Proceedings of the Fourth European Conference on Universal Multiservice Networks*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 419–424.

[64] M. Cheaito, R. Laborde, F. Barrere, and A. Benzekri, "An extensible xacml authorization decision engine for context aware applications," in *Pervasive Computing (JCPC), 2009 Joint Conferences on*, December 2009, pp. 377 –382.

[65] K. K. Konrad, T. Konrad, D. David, S. Howard, and D. Trevor, *Activity Zones for Context-Aware Computing*, ser. Lecture Notes in Computer Science, U. . U. Computing, Ed. Springer Berlin / Heidelberg, October 2003, vol. 2864.

[66] F. Meneses and A. Moreira, "A flexible location-context representation," *the 15th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications, 2004. PIMRC 2004.*, vol. 2, pp. 1065–1069, September 2004.

[67] J. Noh and K. M. Lee, "Application of multiattribute decision-making methods for the determination of relative significance factor of impact categories," *Environmental Management*, vol. 31, no. 5, pp. 0633–0641, May 2003.

[68] H. Barron, "Selecting a best multiattribute alternative with partial information about attribute weights," *Acta Psychologica*, vol. 80, pp. 91–103, 1992.

[69] W. G. Stillwell, D. A. Seaver, and W. Edwards, "A comparison of weight approximation techniques in multiattribute utility decision making," *Organizational Behavior and Human Performance*, vol. 28, no. 1, pp. 62 – 77, 1981. [Online]. Available: http://www.sciencedirect.com/science/article/B7J20-4D5WKJR-25/2/36804352e8836b004795c4f6ddbd5189

[70] H. Barron and B. Barrett, "Decision quality using ranked attribute weights," *Management Science*, vol. 42, no. 11, pp. 1515–1523, 1996.

[71] B. S. A. andKyung Sam Park, "Comparing methods for multiattribute decision making with ordinal weights," *Computers & Operations Research, Part Special Issue: Algorithms and Computational Methods in Feasibility and Infeasibility*, vol. 35, no. 5, pp. 1660–1670, May 2008.

[72] J. Pérez, J. Jimeno, and E. Mokotoff, "Another potential shortcoming of ahp," *Sociedad de Estad stica e Investigaci6n Operutiva (Top)*, vol. 14, no. 1, pp. 99–111, June 2006.

[73] F. Ghotb and lewis Warren, "A case study comparison of the analytic hierarchy process and a fuzzy decision methodology," *The Engineering Economist*, vol. 40, no. 3, pp. 233 – 246, 1995.

[74] S. Giles and D. Bersinic, *MCSA Windows Server 2003 All-in-One Exam Guide (Exams 70-270,70-290,70-291)*. McGraw-Hill Osborne Media, 2003.

[75] A. Ranganathan, J. Al-Muhtadi, and R. H. Campbell, "Reasoning about uncertain contexts in pervasive computing environments," *IEEE Pervasive Computing*, vol. 3, no. 2, pp. 62–70, 2004.

[76] L. Yao and N. Zhang, "A generic authentication loa derivation models," in *Emerging Challenges for Security, Privacy and Trust*, ser. IFIP Advances in Information and Communication Technology, D. Gritzalis and J. Lopez, Eds. Springer Boston, 2009, vol. 297, pp. 98–108, 10.1007/978-3-642-01244-0_9. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-01244-0_9

[77] O. XACML, "extensible access control markup language (xacml) v2.0," http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml, 2005.

[78] D. W. Chadwick, S. Otenko, and T. A. Nguyen, "Adding support to xacml for multi-domain user to user dynamic delegation of authority," *nternational Journal of Information Security*, vol. 8, no. 2, pp. 137–152, February 2009.

[79] M. A. Harrison, W. L. Ruzzo, and J. D. Ullman, "Protection in operating systems," *Commun. ACM*, vol. 19, no. 8, pp. 461–471, 1976.

[80] J. Trent and T. J. E., "Practical safety in flexible access control models," *ACM Trans. Inf. Syst. Secur.*, vol. 4, no. 2, pp. 158–190, 2001.

[81] D. E. Denning, "A lattice model of secure information flow," *Commun. ACM*, vol. 19, no. 5, pp. 236–243, 1976.

[82] R. S. Sandhu, "Lattice-based access control models," *Computer*, vol. 26, no. 11, pp. 9–19, 1993.

[83] A. Saldhana, "Jboss xacml," http://jboss.org/jbosssecurity/downloads/JBoss%20XACML/, June 2009.

[84] Sun, "Sun microsystems' saxparser," http://java.sun.com/j2se/1.4.2/docs/api/javax/xml/parsers/SAXParser.html.

[85] B. Matthias, D. Schahram, and R. Florian, "A survey on context-aware systems," *Int. J. Ad Hoc Ubiquitous Comput.*, vol. 2, no. 4, pp. 263–277, 2007.