ACCESS CONTROL FOR ACTIVE SPACES

BY

GEETANJALI SAMPEMANE

B.Tech., Indian Institute of Technology Bombay, 1991
M.S., University of Illinois at Urbana-Champaign, 2001

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2005

Urbana, Illinois

Abstract

Recent advances in embedded computing and communications technology have facilitated the development of intelligent environments, enabling exciting new applications, but also creating new challenges for security. The large number of heterogeneous devices, mobile users, and new kinds of applications all contribute to making security administration and enforcement more difficult.

We study the problem of *access control* for such environments, which we call Active Spaces. Context plays an important role in these systems—users may have different permissions in different situations, making access control harder to configure, enforce and understand. Collaboration between users is common in these spaces, and needs to be supported by the system.

My thesis is that existing models for access control, such as Role-Based Access Control, can be extended to satisfy the access control requirements for Active Spaces. An access control architecture for Active Spaces must integrate physical and virtual aspects of the environment, provide explicit support for collaborative applications, and support the dynamic and heterogeneous nature of ubiquitous computing environments. Usability, for end-users as well as security administrators, is an important concern. The system must be flexible enough to support a variety of access control models, as new applications, with varying security requirements, are still being developed for these environments.

We propose an access control model that is designed for such environments. We have developed a prototype implementation in the framework of the Gaia system, which we use to demonstrate our thesis. Our model supports discretionary and mandatory access controls, and allows a variety of collaborative modes of usage.

We evaluate our model on the criteria of expressiveness, performance and usability. The model is sufficiently expressive for applications used in our Active Space. The performance overhead of our access control is demonstrated to be negligible. We conducted user studies for an administrative tool for our access control system to identify requirements for security administrative tools. To improve usability for the end-users of this system, we developed *Know*, a framework to provide feedback about access control decisions while protecting the access control policy.

To my parents,
Girija and Venkatram.

Acknowledgements

I would like to thank everyone who supported me intellectually, socially and/or financially during the many years it took me to complete graduate school.

My advisor, Prof. Roy Campbell, and my thesis committee provided support and encouragement during this endeavour. I am grateful to my colleagues and collaborators for many interesting and educational discussions over the years. Especial thanks to Sariel Har-Peled and JD Vaidya for reading early drafts of this document and pointing out where I was being more incomprehensible than usual. Anda Ohlsson was most helpful in solving all scheduling and administrative problems. I also want to thank NSF for funding this research[1].

I would also like to thank Prof. Andrew Chien and my fellow-graduate students from CSAG, who helped me get started with research in Computer Science and taught me to have fun with it despite all the recalcitrant hardware and software.

Lastly, I would like to thank my family for their patience and understanding during my apparently-interminable spell in graduate school.

[1]NSF CCR 0086094 ITR

Table of Contents

List of Tables

List of Figures

List of Abbreviations

ACS	Access Control Service
AL	Access List
CL	Capability List
CORBA	Common Object Request Broker Architecture
DAC	Discretionary Access Control
IDL	Interface Definition Language
IR	Infra-red
MAC	Mandatory Access Control
ORB	Object Request Broker
OMG	Object Management Group
PDA	Personal Digital Assistant
RBAC	Role-Based Access Control
RFID	Radio Frequency Identification
XML	Extensible Markup Language

1 Introduction

Recent advances in computing, communication, and sensor technology have led to the creation of environments that contain a plethora of heterogeneous networked devices. This brings the dream of "ubiquitous computing" or "pervasive computing" closer to reality. Ubiquitous computing envisions a world where large numbers of inexpensive computing devices provide new functionality, enhance user productivity and ease everyday tasks. The goal is anytime/anywhere access to information, with the computing infrastructure becoming ubiquitous enough to blend into the background and no longer be noticed. In homes, offices, and public spaces, ubiquitous computers will unobtrusively augment work or recreational activities with information technology that optimizes the environment for people's needs.

The goal of ubiquitous computing research is to create a user-centric and application-oriented computing environment—users do not want to program a VCR to select the right devices for input and output, they just want to watch a film. Devices in these environments may communicate through a spectrum of networking technologies—from point-to-point infrared, through very local-area wireless communications and body networks, to local-area wireless and wired networks, going all the way to wide-area networks like the Internet and cellular communication networks. This heterogeneous and device-rich environment makes it essential to have some software to integrate all these devices into a computing environment. Developing infrastructure services and applications for such environments is an active research area [DSA99; JFW02; RHC+02].

The Gaia project [Gai00; CHRC01] takes an operating system approach to the problem and provides various services, such as data storage, context, location, and authentication, as middleware system services. The objective is for Gaia to deal with the specifics of interacting with the various devices, just as an operating system on a desktop computer deals with the specifics of interacting with peripheral hardware. Thus, Gaia converts the physical space with the users and devices in it into a unified programmable entity. We call such an environment—the physical space with the devices and appliances in it, and the software services to facilitate interaction with them—an *Active Space*.

Such environments fundamentally change the nature of personal computing. The integration of the physical space with the hardware and software environment facilitates interactive information exchange between users and the space, enabling new types of collaborative applications as well as augmenting traditional models of human–computer interaction. The availability of inexpensive computing devices and sensors and the prevalence of wireless networks are making such spaces increasingly common. A user is no longer logged into a single personal computer, but interacts with a variety of computing devices in the space around him. The set of devices and users is very dynamic, since users and devices are mobile. Applications follow their users, and use the most appropriate devices available in the environment. Flexibility and reconfigurability are important aspects of such systems. The same space is typically used for different applications at different times and by different users. Contextual information, such as the current users in the space, or the current activity, is important for the configuration of Active Spaces.

However, these pervasive computing environments pose new security challenges which must be addressed before they can be deployed for real-world applications. While anytime/anywhere data access is very useful, it can also make it easier for data to fall into the wrong hands. The pervasive nature of these environments increases the risks—for example, enabling criminals to access sensor information from a "smart" home to plan their burglaries better, or stalkers to spy on their victims by getting information from the sensors in their "smart" environments. User privacy is an important concern as the system knows ever-increasing amounts of information about the user. New types of applications and new modes of user interaction with the computing environment (such as using voice or gestures) make it necessary to rethink the traditional user-password approach for computer security. The dynamic environment and the number, heterogeneity and mobility of devices and users makes security management difficult.

Security is often regarded as an obstruction while these systems are still being used for research purposes; the interesting challenge is to connect all these devices to each other in a useful way. However, without effective ways to restrict access to authorized users, these systems can never be deployed for serious applications. Prior experience has shown that it is nearly impossible to retrofit security into systems that have not been designed with security considerations in mind. We argue that the software infrastructure for these environments must contain adequate mechanisms for enforcing the variety of security policies that applications may require. To this end, we study the access control requirements for such ubiquitous environments in this thesis research.

1.1 Problem

The problem I address is that of access control for Active Spaces. Access control is a basic security requirement for guaranteeing security properties such as confidentiality, integrity or availability. Access controls are necessary wherever controlled sharing of resources between different users and applications is desirable.

While access control is a much-studied problem, existing approaches were mostly developed in the context of traditional distributed and multi-user systems, and do not adequately address the requirements of Active Spaces. Access control systems typically specify the allowed accesses by a set of users or subjects to a set of resources or objects. In traditional systems, this set of subjects and objects is reasonably static, whereas in an Active Space, both subjects and objects may enter and leave the space dynamically. An Active Space can also be configured to support different activities, each of which has its own security requirements. Security mechanisms for the space must support this dynamic reconfiguration of the security policies.

While the operating system on traditional multi-user systems isolates users from each other so that their activities do not affect each other, this is not always possible when users are in the same physical space. For example, a meeting being conducted in an Active Space is accessible to all users in the room, and the only way of restricting access to it is to restrict access to the space for the duration of this activity. Thus, a user's permissions to the Active Space are affected by what other users are doing, and by the space *context* in general. An access control system for Active Spaces must consider the physical and virtual context of the space while enforcing security policies.

Active Spaces are commonly used for collaborative applications, where a group of users co-operate to complete a task. Security policies for these spaces must support such collaboration in a secure manner. If the system cannot support such applications, users bypass the security mechanisms to achieve their goals, for example, by sharing passwords to share permissions to a set of files. The system must support limited sharing of permissions (for the collaborative task) between dynamically formed groups of users, where preconfiguring group permissions is not feasible.

Active Spaces are also envisioned to be used for non-technical applications, such as smart homes. Users in these environments may not undergo any special training and expect the devices to be "easy-to-use". Usability considerations such as the "principle of least astonishment" are indicated in such environments. This applies equally to the design of the security mechanisms. While security threats may not

be correctly perceived by users, inconvenient security mechanisms are likely to be disabled. The design of the security systems must be informed by experience from human-computer interaction (HCI) research.

The role-based access control (RBAC) model has become popular in recent years for its ability to support a variety of access control policy configurations (such as mandatory and discretionary access controls). However, the RBAC model as proposed assumes a relatively static environment in which security policies are configured and enforced.

This thesis states that the RBAC model can be extended to support the access control requirements of Active Space applications. The key features that are required in such environments are: explicit support for collaborative applications, integration of physical context into the access control system, and support for dynamic, heterogeneous environments.

1.2 Example

We illustrate the problem with the help of an example. Our example Active Space is a "smart room" in a University. The room contains computers on wired and wireless networks, display walls, microphones, speakers, cameras, and other sensors for light and temperature. Users also bring in personal equipment such as laptop computers or PDAs, which they can use in conjunction with the equipment in the space. This environment is shared by a number of research groups, who use the equipment for various projects. Security policies in this environment are, therefore, context-dependent—the same users have different permissions under different conditions in the space. We describe some of the activities that are conducted in this environment to highlight their requirements pertaining to access control.

The "smart room" is used for a variety of activities, some experimental and research-oriented, and some "production" activities such as seminars and meetings. The room is also often used to demonstrate research to visitors. These activities differ widely in their security requirements. Some meetings may need to be restricted to members of a particular research group, while other seminars may be open to any one in the department. Authorized users must be allowed to participate in these activities, and unauthorized users disallowed from using the space during these activities. This requires that access control policies for the space be reconfigurable on a per-activity basis. Since applications for these systems are still being developed, it is important that the security systems be flexible enough to cope with emerging requirements.

Active Space environments tend to be somewhat decentralized, with the space used by different groups of users. This requires the ability to combine the security requirements of the various parties involved. Mandatory access control policies for the room are required to share equipment securely between the different authorized users. This can be used to specify requirements for room usage that have to be followed by all users, for example, ordinary (non-administrative) users are not allowed to disable system logs. However, personal devices brought in by users need to interact with other users and devices in the room, but their owners will need the ability to control access to them. This is achievable using discretionary access controls. Similarly, users may have privacy requirements that may conflict with the room audit policies, and this needs to be detected and flagged.

User permissions may change during the course of an activity. For example, only the presenter at a seminar has control of the slide projector, whereas the other participants only have read access to the slides. The presenter can use different mechanisms to control the slide projector (e.g. controlling the slides via a cellphone or a PDA, or by using voice or gestures), but other users in the space should not be allowed to do so. Different users may take the role of the presenter at different times, and obtain the associated permissions. Similarly, different speakers during the course of a meeting or class may need access to particular resources within the space. While socially acceptable protocols for passing control of devices may be sufficient in some situations (like a meeting chair calling upon different participants who have questions), access control mechanisms are required for other activities, such as ensuring that only students registered for a particular class are taking the examination.

The physical location of users and devices in the space may also affect their permissions. Many of the Active Space applications are restricted to users who are physically in the room. This imposes two requirements on the access control system—first, that it know who is in the space and be able to restrict access accordingly, and second, that the policies may be reconfigured if and when this context changes. For example, all users in the space can view the contents of the display walls, so it cannot be used to display information that some of the users in the room are not authorized to view. Similarly, a confidential meeting may not be allowed to start while unauthorized users are in the room.

As in the example above, the presence of unauthorized users can change the activities that are allowed in the space. Similarly, users of the room may be allowed to play music on the loudspeakers as long as there are no other users in the room. Again, a user's permissions are affected by the *presence* of other users in the space. This is different from access control in traditional multi-user systems, where a

user's permissions are not affected by who else is logged into the system.

Fine-grained location-tracking of users can also allow permissions to "follow the user" around the space. A user can walk around the room and use any of the various touch-screen displays, and the system will know who is making these requests, since it knows the location of users. Without such fine-grained location-tracking, it may not always be possible for the space to identify which of the users in the room is interacting with a particular device. However, for some activities, it may not always be necessary to identify the exact user. For example, it may be sufficient to know that only a meeting participant is allowed to write to the whiteboard while the meeting is in progress. This is relevant since most of the activities conducted in this room are by groups of users rather than individuals.

There are different types of group activities conducted amongst users with differing levels of mutual trust. Participants in a meeting may all have the same permissions, and shared access to documents created for the meeting. However, a professor teaching a class will have more permissions than the students in the class. Students will not typically be using the whiteboard during a lecture unless asked to by the professor. In other activities, users may obtain permissions for certain activities as part of a group, such as being a member of a committee. The access control system must recognize and support these varying requirements.

While designing security systems for this environment, especially in research environments, it is easy to ignore the complexity of the configuration. However, configuration complexity is most likely to result in poor security in practice. Systems that are too complex to configure correctly are typically left in default modes, or have the security settings disabled. Thus, administrative usability is an important concern for such a system.

The example shows how access control decisions in such an environment are affected by contextual conditions, both physical and virtual. We also described some of the types of user collaboration in the space that we would like to support. Our objective is to design an access control model that can address these requirements. We describe our approach in the following section.

1.3 Approach

To demonstrate the thesis, we proposed an access control model that addresses the special requirements for such Active Space environments. We implemented a system based on this model in the framework of Gaia, a ubiquitous computing sys-

tem for Active Spaces. We then evaluated this system on the basis of expressivity, performance and usability.

Expressivity was evaluated in terms of the ability to express the security policies required for this environment. The basic requirements are to incorporate contextual information into the access control policies, to support discretionary and mandatory access controls and to support various collaborative modes of applications. The access control model extends the role-based access control (RBAC) model to provide the additional features. Group permissions for collaboration are supported by means of "space modes".

The main performance requirement for the access control system was that it not be a bottleneck. To verify this, we tested the access control system with a varying load, and observed its behavior under rapid changes of context and increasing numbers of users and devices. Detailed performance results are presented; however, the main result is that the delay introduced by access control was insignificant. This is an important concern, since security mechanisms that are perceived as introducing unacceptably large performance overheads are often disabled in practice.

Usability evaluation is harder, since there are no clear metrics. We consider two aspects of usability: for the security administrator, who has to configure the access control policy, and for the end-user, who is faced with the decision of the access control system. We performed user studies on an administrative tool designed for this interface, from which we conclude that the configuration complexity is within acceptable limits. The study also identified some requirements for security administration tools. We improved usability for the end-user by means of the *Know* framework, which provided feedback about the access control decision to users who were denied access. The reasoning behind this was that users in such systems are often denied access because of changes in context that are hard to perceive, and that the system has enough information about the user to provide helpful information.

1.4 Thesis contributions

The result of this thesis research is a demonstration of an access control system for a ubiquitous computing environment. The specific contributions of this work are described below.

Access control model for ubiquitous computing environments: The model supports policies that can represent the physical and virtual context of the space.

The model is flexible enough to support DAC and MAC policies.

System implementation and evaluation: The implementation of the model within the Gaia system demonstrates the practicality of the system. In particular, the low performance overhead makes it feasible for use within Active Spaces.

Collaborative environments: We identify the different types of collaboration common in Active Space environments and provide access control support for the differing levels of mutual trust between users in such environments.

Administrative usability: A user study of a tool for security administrators who need to configure this system identified some general requirements for security administration tools.

Feedback for access control: The *Know* system provides a framework for feedback about access control decisions to improve system usability while still honoring meta-policies to protect policy confidentiality.

1.5 Roadmap

The rest of this document is structured as follows: Chapter 2 provides the necessary background required to understand this work before we provide the problem statement in Chapter 3. Chapter 4 describes the access control model and the policy specification. The system design and implementation are presented in Chapter 5, along with a performance evaluation. We describe the usability issues considered during system design in Chapter 6. We conclude in Chapter 7 with a discussion of future work.

2 Background and Related Work

This chapter explains the terms that are needed to understand this research and describes related research. We explain the basic access control problem in Section 2.1. Then, in Section 2.2, we describe the ubiquitous computing environment and the Gaia system that we use to demonstrate the thesis research. Section 2.3 describes previous work in access control in collaborative environments, while Section 2.4 describes work on usability and security. Section 2.5 describes some other related work. Finally, in Section 2.6, we outline our approach to the problem in relation to existing work.

2.1 Access control

The basic access control problem is to ensure that all accesses to system resources are authorized. An access control system receives requests of the form "Can a user U perform an action A on an object (or system resource) O?" and responds with a "Yes" or a "No".

Access controls are essential to regulate any sharing of resources. They are a requirement of any system that wants to provide any of the security properties such as confidentiality, integrity or availability.

The problem of designing access controls, which are also known as protection mechanisms, has been studied since the early days of computing [Lam71]. Computers started out having restricted physical access and only authorized users were allowed to submit programs. Even in such a system, access controls can restrict what parts of the system a user program is allowed to access. Multi-tasking systems require that different processes (even those belonging to the same user) be protected from each other to prevent one process from accidentally or maliciously harming another. Multi-user systems need to protect processes belonging to different users from harming each other. This is typically handled by authenticating users when they login and creating separate protection domains for them. Networked systems allow remote users to access resources, and various mechanisms, typically based on requester identity or network location, are used to restrict this

Object / Subject	S_1	S_2	S_3	O_1	O_2
S_1	own read write			own read	read
S_2		own read		read	own write
S_3			own read	read	write

Figure 2.1: Access Matrix

access. A typical operating system may employ different techniques to restrict access to different resources to authorized users or processes.

An access control system consists of a security *policy* to specify authorized accesses and *mechanisms* to enforce this policy. Different policies are required for different access requirements. Policies and mechanisms are generally studied separately, but in a practical system, a policy must be enforceable by the available mechanisms. Saltzer and Schroeder [SS75] identified a set of design principles for information protection mechanisms.

An access control request has three parameters: a *subject* making the access request (which could be a user or program on behalf of a user), a system *object*, and the specific object *right* (such as read, write, execute, or the ability to call a method on an object) being requested. The effectiveness of access controls depends on proper (and unforgeable) user (subject) identification and protection of the policy from unauthorized modification.

One of the earliest models for representing access control systems is the access matrix model, proposed by Lampson [Lam71], and extended by Graham and Denning [GD72]. The model is defined by states and state transitions, where the protection state of the system is represented by a matrix and the transitions are described by commands that modify this matrix. Subjects in the system are represented as rows of the access matrix and objects as columns. Each cell in this matrix contains the rights of a subject on an object, as shown in Figure 2.1.

An authorized state is an access matrix that contains only authorized rights. An access control policy (also called a protection policy) partitions the set of all possible states into authorized and unauthorized states. The state can be modified by commands that create or destroy subjects or objects, or add or remove rights from the access matrix. In practical access control systems, different sets of users may be allowed (or authorized) to perform these commands. A policy specification is

secure if the set of all reachable states are authorized, starting at the given initial (authorized) state [Den82]. Alternatively, a secure access control system obeys the following safety property: the access matrix at any point of time contains only authorized access rights. Harrison et al. [HRU76] proved that in the general abstract case, the security of computer systems was undecidable, and explored some of the limits of the system.

Access control policies can be classified based on who is allowed to change the access matrix. Most operating systems today implement some form of *Discretionary Access Control* (DAC), where the *owner* of an object (generally its creator) is allowed to permit or deny other users access to it. A problem with DAC is that it is hard to enforce a system-wide policy, since the owners of the individual objects can affect the policy. Instead, military systems have tended to use *Mandatory Access Control* (MAC), where the owner of an object cannot change its security attributes. A security administrator sets the access policy, and a system mechanism enforces the access control policy. An example of a MAC system is one in which documents (objects) are classified into categories such as "top secret" or "confidential" and users (subjects) with appropriate security clearances have access to them, but cannot change the document classifications.

The access matrix is useful as an abstract representation of the protection state of a system. It is rarely used to implement a protection system, because, in practice, it is large and sparse. Access rights to objects are traditionally stored in access lists (ALs) or capability lists (CLs). Each object has an AL, which contains a list of subjects and their access rights to this object. Alternatively, each subject may have a CL, which contains a list of objects and rights that it is allowed. Conceptually, merging these two sets of lists would result in the access matrix.

Around the same time that the access matrix was being used to model protection in operating systems, Conway et al. [CMM72] modeled protection in database systems by means of a security matrix. Here, the subjects are users and the objects are files, records or other data objects. Entries in the matrix are decision rules that specify the conditions under which a subject has access to a data object. This allowed a limited form of "context-dependent" permissions—permissions could be data-dependent (based on the values of the data item being accessed), or time-dependent, or based on access history (what other documents the subject has accessed so far).

2.1.1 Role-based access control (RBAC)

One of the most popular access control models in recent years has been role-based access control (RBAC), which is based on the principle that access control decisions are based on the roles that individuals take on as part of an organization [SCFY96; FK92]. The key concept in RBAC is a role, which is associated with a set of permissions. Roles may be organized into a hierarchy to represent organizational hierarchy. RBAC maintains two mappings: a User-Role Assignment (URA) and a Role-Permission Assignment (PRA). These two mappings can be updated independently. Users can be added to the URA when they are to perform a new function. The key insight in RBAC is that the URA and PRA change less frequently than permissions of individual users.

The RBAC family contains a family of four models [SCFY96], known as RBAC_0, RBAC_1, RBAC_2 and RBAC_3, that share the basic idea, but have different additional features. RBAC_0 is the basic role-based access control model, consisting of the following components:

- U, R, P and S, sets of users, roles, permissions and sessions, respectively
- $PRA \subseteq P \times R$, a many-to-many permission to role assignment relation
- $URA \subseteq U \times R$, a many-to-many user to role assignment relation,
- $user : S \rightarrow U$, a function mapping each session s_i to a single user which is constant for the session lifetime, and
- $roles : S \rightarrow 2^R$, a function mapping each session to a set of roles, $roles(s_i) \subseteq \{r|(user(s_i), r) \in URA\}$, which can change with time.

Role hierarchies are introduced in RBAC_1. These allow roles to be structured to reflect an organization's lines of authority. Thus, RBAC_1 introduces, in addition to RBAC_0, a partial order RH on the roles, written as $\geq$, and known as the role hierarchy.

The next version, RBAC_2, introduces the concept of constraints [AS00], which can apply either to the URA or the PRA relation of RBAC_0, or to the $user$ and $roles$ functions. They are predicates that, when applied to these relations and functions, return a value of "acceptable" or "not acceptable". While the model allows for arbitrary constraints, implementation considerations generally limit the complexity of constraints that can be checked and enforced. Constraints are a useful mechanism for specifying higher-level organizational policy such as "separation of duty". They enable more dynamic aspects of the environment to be incorporated in access control, but implementation support for constraints has been limited.

Role hierarchies *and* constraints are provided in RBAC_3, which combines RBAC_1 and RBAC_2.

One of the goals of RBAC is to simplify administration. If roles map directly onto organizational functions, it is easy to represent the organization security policy using RBAC. An administrative model for RBAC [SBM99] has been proposed, with a separate hierarchy of administrative roles. RBAC is policy-neutral and has been used to model both MAC (where roles represent security "levels" or clearances) and DAC (where roles represent identities) systems [NO95; OSM00].

One of the problems with RBAC, however, is that it does not directly allow for the incorporation of anything other than user identity (or role membership) into the access control decisions. Constraints allow for the specification of more complex policy conditions, but most implementations of RBAC do not support complex constraints. Various extensions have been developed to allow other information to be incorporated into the RBAC model, some of which are described below.

RBAC has become popular in recent years, and has been extended in various ways for different applications. It has been used to model workflow authorization [BFA99]. Bertino et al. [BBF01] propose a model to incorporate temporal considerations into RBAC. They do this by supporting periodic triggers that can enable or disable role activation on a temporal basis.

Kumar et al. [KKC02] describe a context-sensitive RBAC, where user contexts and object contexts can be used to create role contexts, which can be specified by administrators during role creation. These role contexts are Boolean expressions that can limit the applicability of a role's permission to a subset of the instances. Object roles, which allow specifying permissions to groups of similar objects, are also described in the GRBAC [CMA00] model. These concepts can be useful for Active Spaces.

Role-based access control has been adapted for ubiquitous computing environments [Gil01; Vis01; CMA00], with the concept of roles extended in various ways to deal with context information. The Aware Home project has extended RBAC to allow context-aware policies, such as those based on temporal authorizations. They do this with the help of object and environment roles [CMA00; CLS+01; CFZA02]. Their work is closest to ours, but we also explicitly address the permissions for collaborative activities in the space, and the effect of users concurrently in the space on each other.

2.2 Ubiquitous computing

Marc Weiser articulated a vision of ubiquitous computing [Wei91] where computation becomes so cheap and small that it is embedded into everyday objects and moves to the periphery of human attention. Recent advances in communication, device miniaturization and sensor technology have brought this vision closer to reality.

Much of the early work in ubiquitous computing focused on simplifying the interaction between humans and their computing environment, leading to the development of applications that allow users to interact with applications in more natural ways than via a keyboard and mouse—using speech or gestures, for example. One of the novel features of such environments was the introduction of context into applications, where context can be anything from the physical world that affects an application—location, time, weather, user mood etc. It soon became apparent that many applications performed similar base operations and some system support for these operations would simplify application development. This led to the development of tools such as the Context Toolkit [DA00] to provide contextual information about the surrounding environment.

Ubiquitous computing tries to bridge the physical and virtual environments by augmenting physical objects with intelligent sensors and incorporating an array of software, hardware and applications into computing environments. These environments consist of intelligent rooms or spaces that contain appliances (such as video walls or whiteboards), computers and users with mobile networked devices. There may be different input-output mechanisms and other sensors to detect contextual conditions like temperature and lighting. Thus, the computational environment is composed of the physical space, along with all the devices and appliances in it. While networked applications have traditionally attempted to hide physical location by providing uniform interfaces for local and remote users to access services [BG99], spatial location is often important to the organization of communication in intelligent environments such as smart rooms [Pen96; HB00].

As these devices have become more numerous and more capable, software infrastructure to manage them has not kept pace. Interconnecting or running applications across a set of such devices still poses a challenge. More recently, various research projects are working on providing infrastructure support for using such interactive workspaces [GSSS02; JFW02; Oxy; RHC+02].

Ubiquitous computing spaces are used in different ways from traditional desktop systems. There is no longer a one-to-one relationship between users and applica-

tions. Spaces are typically used as shared workspaces to support group activity. Applications in the space typically run across a variety of heterogeneous devices, and can be dynamically reconfigured.

2.2.1 Active Spaces and Gaia

The Gaia project focuses on developing infrastructure services for a class of ubiquitous computing environments called Active Spaces. As physical spaces are integrated with the hardware and software they contain, they become interactive and programmable, turning into *Active Spaces*. Homes, offices, classrooms and public spaces may all turn into Active Spaces, but they share some common defining criteria:

Heterogeneity: The environment typically has many computing devices, of varying capabilities and resource requirements.

Mobility: Users and devices in this environment are mobile. Applications and data can move with the users.

Distributed nature: Applications use the resources in the space, and may be partitioned to run across different devices.

The Gaia approach is to treat the Active Space itself as the computing environment. The software approach is analogous to an operating system for a desktop computer—just as the operating system on a desktop computer manages the computing resources, input/output devices and peripherals, the Gaia "OS" manages the details of interacting with devices in the Active Space and provides application programmers a uniform interface to develop to. Gaia acts as a "meta-OS" for the space and provides virtualized access to resources. Middleware services in Gaia provide infrastructure services, including a filesystem [HC03; Hes03], events, naming, context and location. An application model [RC03; Rom03] allows developers to create generic applications for an Active Space without needing to know the specific hardware configuration of a space. When the application is instantiated, the devices required by the application are mapped onto currently available resources in the space.

Gaia provides a set of kernel services. The general system architecture is shown in Figure 2.2. We describe some of the important services below.

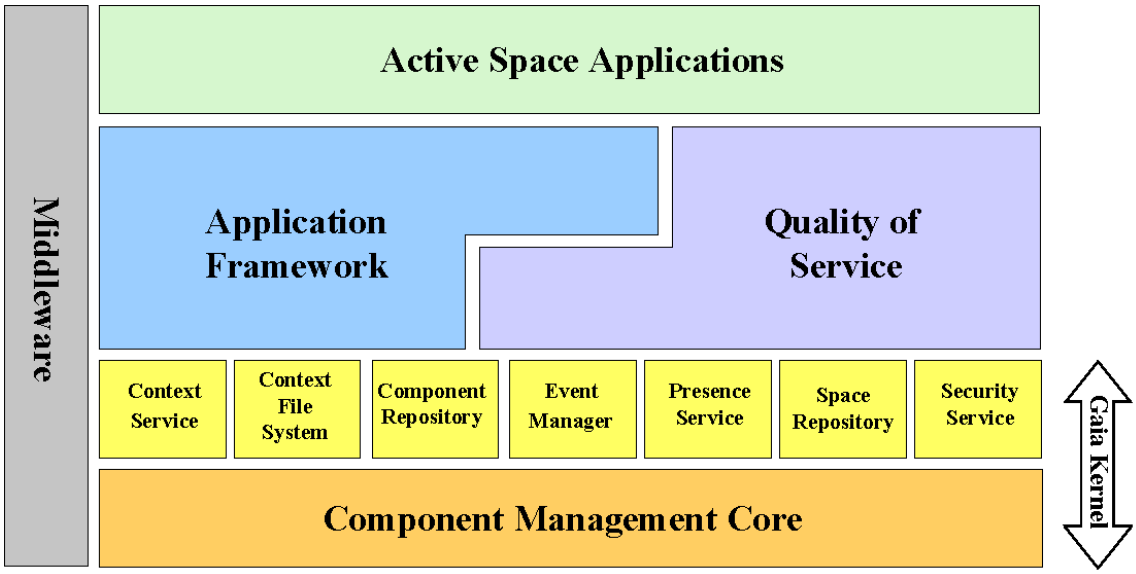


Figure 2.2: Gaia kernel services

Gaia Component Model

The *Component Management Core*, which is the lowest layer in Gaia, manages component instantiation and destruction. This core acts as a remote execution node, and can be instructed to load and run Gaia components. Each host that joins a Gaia Active Space runs an instance of the Component Management Core, which is implemented as a service known as the UOBHost. Gaia applications are typically a set of components that run on a set of networked hosts. When components are instantiated, they register into the Gaia system, and can be contacted by other components in the system. In the current implementation, Gaia components are typically implemented as CORBA objects.

Space Repository

The Space Repository maintains information about all the hardware and software entities in an Active Space. Services, applications and devices are all registered in the Space Repository when they start up in an Active Space. Users known to the Authentication Service are also listed in the Space Repository. Thus the Space Repository maintains information about the run-time state of the space. Figure 2.3 shows the contents of the Space Repository in an Active Space.

The Space Repository also maintains more static information about the characteristics of the entities (users, devices, services and applications) in the space. These are

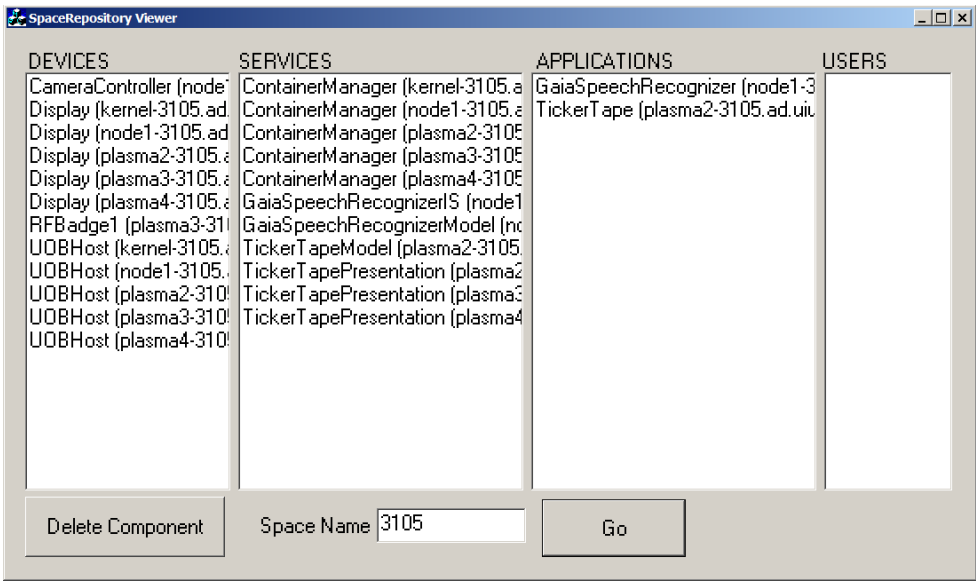


Figure 2.3: Snapshot of the SpaceRepository

provided in the form of XML files containing attributes describing these entities.

Applications contact the Space Repository to get information about available resources in the space. This allows applications to be developed in a generic manner and then be mapped to the specific resources available in a particular Active Space.

Application Framework

The Gaia Application Framework [Rom03] provides an interface based on the Model-View-Controller [KP88] pattern. Gaia applications are partitioned among a set of co-ordinated devices, receive input events from different devices, present their state using different types of devices (such as audio, video display or by controlling the temperature) and can adapt to changes in the environment. Applications consist of a set of components: a model, one or more presentations, one or more inputsensors and a co-ordinator. The model contains the application logic. Presentations are different "views" of the output, and the same application may use different presentations based on the facilities available in an Active Space. Inputsensors allow users to interact with the application. Again, different types of inputsensors may be used for the same application. A co-ordinator maintains the link between these components, and allows new presentations and inputsensors to be dynamically added, moved or deleted during the application lifetime.

Authentication Service

The Gaia Authentication service [AMRCM03] identifies users within a space, using a variety of techniques, such as biometrics, badges or passwords. Different methods of authentication may be trusted to different extents. The Authentication Service maintains a list of users who are logged into a Gaia Active Space. This information may be used to configure the access control policies for the space. The Authentication Service can issue credentials to authenticated users, and these credentials may be used by the access control system to verify that users requesting services from the space are authorized to receive them. We describe the issues related to using these credentials, and more details of the interaction between the Authentication Service and the Access Control System in Chapter 5.

Location Service

An Active Space may contain a variety of location-tracking devices, with varying physical properties. RFID badges can provide approximate location information, while UbiSense tags can be tracked with a much finer granularity. The Gaia Location Service combines information from all these sensors and provides location information to users and applications. The Location Service also provides an interface that applications can use to define regions, query for the location of particular objects, or to track objects as they enter and leave regions of interest.

Location information can be important for access control in Active Spaces. Users may be permitted access to some resources based on their location. User location can also be used to identify the source of requests—a request made from a touchscreen display can be recognized as coming from the user standing in front of it. For other applications, access to resources may be permitted to users who are "near" something, for example, all users *within* a building may be allowed access to the calendar of events occurring in the building that day.

Context Service

Since Active Spaces are heavily context-driven, Gaia needs a mechanism to sense context information from the environment. A Context Service, consisting of context providers, provides this information in Gaia. All resources in Gaia are accessed through middleware services. Services also provide access to devices within an Active Space. Thus, controlling access to resources in Gaia requires mediating access to these services.

2.2.2 Security in ubiquitous computing

While considerable research has gone into adapting traditional applications to an environment of heterogeneous devices [WKJ+01], the security issues have not received much attention so far. There has been work on authentication and trust in ubiquitous environments, but not much attention has been paid to the problem of authorizing users in such dynamic and heterogeneous environments.

The Aware Home project [AHR] has developed context-based security mechanisms for such environments. Generalized RBAC [CMA00] extends the standard RBAC system with object and environment roles, which can be used to represent some contextual information. Zhang and Parasher [ZP04] have proposed an extension to RBAC that activates some subset of user roles based on system context provided by a Context Agent.

Kagal et al. [KFJ01] propose a trust-based delegation model for pervasive computing environments. Entities can delegate trust to third-parties. For example, the user of an office can allow limited office access to a visitor, even though the security policy does not specify access to visitors.

Shankar and Balfanz [SB02] propose a scheme where contextual information about the user making the request (such as the location) can be used to automate security management.

Bullock et al. [BB94] describe an approach to access control for collaborative virtual environments based on access to physical spaces. Access to an object depends on where it is currently located and who has access to that space. They propose that groups of users could obtain permissions that can allow them to enter certain spaces. They do not address the issue of how to form or identify such groups.

Another security concern in such environments is privacy. Since these systems often collect more information about their users than traditional computers, by means of pervasive sensors and the like, they can pose a bigger threat to user privacy if the information collection and data handling systems are not designed carefully. Marc Langheinrich [Lan01] highlights the importance of privacy considerations in the design of ubiquitous computing systems. Another area of concern with respect to privacy is the prevalence of capture applications, in which audio, video or other sensors record copious information about everyday environments. A common application in pervasive computing systems is location-tracking and the provision of location-based services. This leads to the problem of controlling access to user location information [HS04].

2.3 Access control in collaborative environments

Active Spaces are typically used as shared workspaces, or by groups of collaborating users. Much research on collaborative systems has been conducted in the Computer Supported Cooperative Work (CSCW) area. Key issues of CSCW are group awareness, multi-user interfaces, concurrency control, communication and co-ordination within the group, shared information space and the support of a heterogenous, open environment which integrates existing single-user applications. Security has not been a major consideration since the environment assumes a relatively small group of co-operating users.

Pettifer et al. [PM01] describe some types of virtual environments (such as simulation of real environments used in training, or fictional game environments) and describe the types of access needed in collaborative virtual environments. They do not discuss enforcement mechanisms, but describe the types of policies that might be useful in such environments, and discuss system support for such policies in the context of an entity/programming architecture called Deva. The key features of these access policies are the context-based conditions and requirements of agreement from other collaborating users (e.g. "I will agree to this after 6pm or if A,B and C all agree").

Jaeger et al. [JP96] outline the requirements for role-based access control for collaborative systems. They argue that RBAC systems need to implement a DAC model to allow collaborative users to control rights to their own objects and allow other users some subset of their rights.

Kang et al. [KPF01] model inter-organizational workflow, pointing out that these typically require fine-grained access control on workflow objects (such as documents). Access may be allowed depending on contextual conditions and dynamic constraints (i.e. a user may have different access to a document based on who has performed some previous task on the document).

Shen et al [SD92] provide a generalized editing model for collaborative access control, whereby users interact with a collaborative application by concurrently editing its data structures.

There has been some work on access control for coalitions [CTWS02] or co-operating organizations. Phillips et al. [PTD02] describe the problem of information sharing and security in Dynamic Coalitions, such as organizations working together temporarily to respond to crises such as natural disasters or wars. Users belonging to these different organizations must be able to access information and perform ac-

tions for the duration of this coalition, but more restrictions may need to be applied, such as temporal constraints on the permissions. Gligor et al. [GKK+02] discuss the negotiation of access control policies among collaborating institutions that want to share some resources for the collaborative task. Khurana et al. [KGL02] propose the formation of coalition authorities to issue authorizations to access coalition resources.

The dRBAC [FPP+02] model is a decentralized trust-management and access control mechanism for systems spanning multiple administrative domains, such as those encountered in grid computing [FKT01].

Task-based Authorization Controls [TS97] model access control requirements from the task's point of view—different permissions are allocated to different users at different stages in the task. Georgiadis et al [GMPT01] used a team-based access control scheme to support fine-grained policies using RBAC. Teams of users can be assigned permissions for a particular task, and some contextual information such as time and location can be considered by the access control system.

The main difference between the above work and ours is that collaborations in our environment may be more ad hoc, such as groups of students working together in a space, and we would like to enable collaboration without requiring administrator intervention.

2.4 Usable security

Usability has been recognized as an important concern for security systems since the early days [SS75] of research in computer protection systems; however, in practice, usability issues have not been a primary consideration for security designers. One reason for this could be that much of the early access control work was performed in the context of military applications, where it is reasonable to assume that users are aware of the importance of security, and are likely to undergo training and follow orders while operating systems. However, this model is not appropriate for the most common use of computers these days, in homes or at work, where users are not properly aware of the risks of security failures, and disable security mechanisms that interfere with what they want to do. Usability concerns are especially important in ubiquitous computing environments, since the objective of ubiquitous computing is to blend into the background and allow the user to perform his tasks without having to pay attention to the computing environment.

As computing becomes more pervasive, the weakest link in the security system is

often the human factor. Many attacks today target legitimate users rather than attempt to break cryptostems. For example, "phishing" attacks that lure users into entering personal information into websites created to steal this information work because it is not obvious to users who they are giving this information to. Similarly, various types of worm/virus attacks depend on users clicking on attachments without knowing exactly what they do. While users are now exhorted not to click on links, more usability considerations while designing these systems would have been a better solution.

Recently, there is increasing interest in the field of HCISEC, which considers the usability or "human-computer interaction" aspects of security mechanisms. A study of PGP [WJ99] found that even when using software designed for security, users often did not have a clear model of what was supposed to happen, and misused the software so that they did not obtain the security it was supposed to provide. The authors identified some guidelines for designing security interfaces. More work in this area was done by Yee [Yee02; Yee03], arguing that the principle of Least Authority is essential to usability of secure systems. Consistent feedback has been identified as an important aspect of usability. This led us to explore options to provide feedback about access control to users of the system.

2.5 Other related work

Bonatti et al. [BdS00; BDS02] propose an algebra of security policies to address composing authorization specifications from different sources (and perhaps, different languages and enforcement mechanisms). They define policies as a set of authorization terms, representing the outcome of a policy specification. This is similar to our approach, where the DAC and MAC policies are composed to result in a set of authorizations that are enforced by the Access Control Service.

Policy hashing [KHJ03] has been proposed to protect the policies for a firewall from less trustworthy enforcement points. This prevents intruders from reading sensitive policies on compromised enforcement points. Feedback to end-users is not a consideration. Access control systems for Web publishing [BDS01] provide more information about the policy if conditions needs to be changed for access. However, policy protection is not addressed. Trust negotiation protocols [BS02; WL04; YWS03] address the problem of protecting the confidentiality of credentials of both parties involved in a session. At each stage, one of the parties must provide feedback to the other party as to what credentials are needed to proceed with the negotiation. These systems are similar to our policy feedback system, *Know*, which can

augment these systems at each stage of trust negotiation by providing useful feedback. With respect to suppressing feedback options, Bonatti et al. [BS02] protect the server's *state* by filtering policy feedback. Such techniques can also be applied to *Know*, which protects the server's *policies*. Policy protection in [BS02] is achieved by progressively revealing more requirements depending on credentials revealed by the user.

2.6 Our approach

To summarize, access control in the context of traditional operating systems has been well-studied, but the novel characteristics of Active Space environments require new facilities. We believe that such ubiquitous computing environments cannot be deployed for serious use until a security architecture is in place. We have developed an access control model that can support context-based access control policies in a collaborative environment, while addressing usability concerns. In addition to supporting present-day applications, the model is designed to be flexible, so that it may satisfy access control requirements of new applications in this rapidly-developing area.

3 Problem Statement

Section 3.1 describes the context in which this work was performed, and the assumptions we make about the system environment. Section 3.2 presents the precise thesis statement and Section 3.3 discusses how to evaluate our approach.

3.1 Environment

Access control deals with the problem of ensuring that all access to resources within a system are authorized. It is impossible to guarantee any of the security properties of confidentiality, integrity and availability without effective access controls. An access control system consists of *policies* that specify the authorized accesses within a system, and *mechanisms* to enforce these policies. While access control has been studied since the early days of shared computing systems, Active Spaces pose some novel challenges. In this section, we identify the specific problems for access control in Active Space environments.

3.1.1 Dynamic environments

Active Spaces are very dynamic. Users, devices and applications are mobile, entering and leaving the space in an unpredictable fashion. Furthermore, physical resources are composed dynamically in a multitude of ways to support user tasks. This mobile and dynamic environment poses many challenges for an access control system.

Dynamic users and devices

Traditionally, access control policies specify the authorized modes of access for a set of *subjects* (or users) to a set of *objects* (or resources) in the system. These sets are typically fairly static. In Active Spaces, both subjects and objects in the system change frequently. Users of the system are not fixed. For example, visitors to a building are a common occurrence and need access to resources in the building.

Similarly, the set of devices in the environment vary as they are carried around by users. In addition to mobility, the same resources may be used for different tasks at different points in time. Adaptability and reconfigurability are important characteristics of the environment. Access control policies must support this dynamism and be able to express this reconfiguration.

Mobile applications

Applications in these environments can follow their users, getting instantiated on different sets of hardware resources. Application policies are thus not attached to any set of devices, but must apply to an instance of the application, and are set by the user. Thus, access control policies need to be more user and application-centric than system-centric.

3.1.2 Context

Context plays an important role in ubiquitous computing environments. Human beings interact with each other based on an implicit shared context, which has not been available so far in interactions with computers. Active Spaces, however, attempt to incorporate context about the physical world and the current activity into the configuration of the space.

Physical context

Active Spaces have a variety of sensors that allow them to obtain contextual information, such as time of day, number of users present, ambient temperature or lighting conditions. Ubiquitous computing often combines the physical world with virtual resources to deliver context-rich applications. Access control is no longer determined strictly by a user's identity and access rights, but also by the physical context. Traditional multi-user operating systems isolate users from each other, so that multiple users on a system do not affect each other. This is not feasible in an Active Space, since the physical aspects of the space affect the virtual ones—multiple users in a room can *all* see the videowall, so it cannot be used for displaying data that they are not all allowed to view. Security architectures for Active Spaces must integrate the physical context with the virtual for access control, taking into account the fact that users cannot always be isolated from each other.

Virtual context

An Active Space may be used for different tasks at different times. Access to devices depends not only on the user, but on the task for which the space is being used. For example, a student in a classroom is not allowed to use a cellphone while a class is in session, but may do so in the same room at other times. As such, an access control architecture must support dynamic decisions based on the virtual context.

Feedback to users

Since user permissions within a space change with context, and that context is often *implicit*, it may not be easy for users to understand why they are now disallowed from doing something they have been able to do in the past. In order to avoid user frustration due to apparently erratic or arbitrary behavior, an access control system must provide good feedback to the user. Since Active Space applications may not have a traditional keyboard and display interface to interact with users, new feedback mechanisms have to be used.

3.1.3 Collaboration

Many Active Space applications involve groups of users working together towards a common objective, requiring members of the group to have shared access to certain resources and information. Access control for these groups of users must be supported. As these groups are often formed in an ad hoc manner, it is not reasonable to expect security administrators to pre-configure them into access control policies.

Support for ad hoc groups

Collaborative applications involve a set of users who wish to share some permissions in a limited manner. For example, a group of students may get together to use an empty classroom to work on a class project. They wish to combine their rights to the files and devices they use for this project, but *not* permissions to all other files and applications. Traditional access control systems have not provided much support for dynamically-created groups of users. Administrators can assign users to groups in common operating systems like Unix or Windows, and assign permissions to these groups, but these groups tend to be relatively static. We would like to

support ad hoc groups of users getting together to collaborate, without requiring administrator support for them to be added into a shared group. On conventional systems, users often resort to potentially insecure methods, such as sharing passwords. There are different types of collaboration, based on the level of mutual trust between group members—an access control service should provide explicit support for these.

3.1.4 Policy management

Managing security in ubiquitous computing is challenging due to the complexity and dynamism of the environment. Tools to aid security administrators in managing this complexity must deal with the large number of devices and multiple sources of security policies.

Scale

The sheer number of users, devices and applications make Active Spaces complex to manage. In particular, security administration is complicated by the large number of users and resources in the space, because it is easy for a misconfiguration to be lost in the volume of information.

Multiple security policies

Active Spaces have multiple sources of security policies. System resources, mobile devices, and applications all have their own system or user–defined policies. An access control service must be able to compose policies in meaningful ways, and provide adequate feedback both to the end user and the security administrators so that access control decisions do not appear arbitrary.

Configuration complexity

Configuring an access control system for real security domains is error-prone. In role-based access control systems, a user may have several roles, and a role may be applied to multiple users. It is difficult for a security administrator to understand all the ramifications of changing any particular role-to-permissions mapping. For example, it is easy to give *more* users access to a resource than intended.

3.1.5 Assumptions

We assume that users are authenticated on entering an Active Space. The space thus knows which users are present in it. The details of the authentication mechanism are not important. Anonymous use of the space may be permitted, but the space must still know of the presence of an "anonymous" user in order to configure policies appropriately. Security policies of a particular space may or may not permit anonymous usage.

When users are authenticated, they are issued a credential by the authentication service. This credential accompanies all requests for service made by a user, and is used by the access control service to check for authorization. We assume that these credentials are unforgeable, and can be securely transmitted within the system.

It is not always possible for a space to distinguish between the different users in it. For example, when a group of users are grouped around a videowall and using a touchscreen to enter data for an application, it is impossible to distinguish between the users without extremely intrusive authentication before every action. The access control system is based on the assumption that it may not always be possible to distinguish which user in the space has performed a particular action.

We assume that users will bring their own devices, such as laptops and hand-held computers, into the Active Space, and will want to restrict access to these devices as they see fit. Hence, we support discretionary access control (DAC) policies for such "user-owned" devices, while the policy for the devices that belong to the space is a mandatory access control (MAC) policy.

Access to devices in the space is mediated through software services. The access control system operates by intercepting requests at this level. Denial-of-service attacks, such as physically disconnecting devices in the space, are not addressed. We assume that access to the space is restricted to authorized users, and this, together with audit mechanisms, can mitigate this threat.

3.1.6 Solution space

We propose a model for access control in Active Space environments that addresses their particular security requirements. However, one of the objectives for our proposed solution is that it must be practical to implement within existing Active Space environments. As these environments are still being used in experimental ways, with new types of applications being proposed, the access control scheme must be flexible enough to support a variety of security requirements, which may change

with more experience with such environments. It is also not reasonable to expect that intrusive security mechanisms will be accepted in an system where the objective is for computing to blend into the environment and become invisible.

Our goal, therefore, is to come up with an access control model that can satisfactorily express policies for Active Space environments, and mechanisms that can implement these policies within existing ubiquitous computing environments. While we can modify the system software in the space to provide support for the necessary mechanisms, we strive to minimize these required modifications to simplify deployment. Applications cannot typically be modified at the source code level, though it may be possible to provide some control of the run-time environment by means of wrappers or launchers. Ideally, the access control system must enforce the space policy, while remaining transparent to the applications.

3.2 Thesis statement

The thesis can be stated as follows:

Access control for ubiquitous computing environments can be provided by extending existing models, such as role-based access control, to incorporate physical and virtual context information into the access control decision and to provide support for collaboration amongst users in the space.

3.3 Success criteria

To demonstrate the thesis, we develop a model for access control for Active Space environments and evaluate it. The evaluation criteria we will use are:

- Expressiveness: Our access control model must be expressive enough to support the requirements of Active Space environments. We study the applications used in such environments to verify that our model can support these.
- Performance: If the overheads imposed by the security mechanisms are too high, users disable them, and run such systems in insecure manners. Performance considerations are therefore important in the design and implementation of practical security systems. We evaluate the performance of our system and demonstrate that it is not a bottleneck in the Gaia environment.
- Usability: Usable security is particularly important in such environments since we have non-expert users, and new applications. The complex environ-

ment poses challenges for administrators who must configure policies that make sense. Incorporating context into the security decision adds another variable, increasing the complexity and making it harder for users to understand what is happening. We provide some tools for administrators to configure the security policies, and design the system to provide feedback in cases where access is denied.

4 Access Control Model

In this chapter, we provide a formal description of our access control model. Section 4.1 identifies the limitations of existing models, and describes the requirements for our model. We describe the model in Section 4.2, provide a formal specification in Section 4.3, and follow it with a validation in Section 4.4. We evaluate this model by discussing its applicability to Active Space applications in Section 4.5.

4.1 Requirements

The dynamic, decentralized and context-dependent nature of Active Spaces make new demands on the Access Control System. While there are many well-studied models for access control, none of them sufficiently satisfy the requirements for Active Spaces.

Role-based Access Control (RBAC) [FK92; SCFY96] is probably the most popular model for access control in recent years, and has been extensively studied. In an RBAC model, roles represent functions within an organization, and permissions are assigned to roles rather than to individual users. Since roles represent functions within an organization, role-based models can support an organization's security policy more naturally. Administration is simplified—a user moving from one department to another does not require all authorizations to be individually revoked and new ones granted, but just needs to be assigned to the new role.

However, in the basic RBAC model, permissions granted to a role are based only on the resources required for the particular function. There is no mechanism for expressing other factors that may influence authorization. In many systems, including Active Spaces, access is restricted based on other attributes of the computing environment, such as time-of-day or the type of computer of network connection being used. While separate roles could be created for each of these possibilities, with role constraints used to control their activation, this leads to a role explosion and an administrative nightmare.

The RBAC model is suitable for assigning permissions to users in an organization, where permissions are relatively static, and permissions assigned to roles do not

change much. The roles are expected to be pre-defined by a security administrator to specify the permissions required for specific organizational tasks. It is not easy to specify permissions in a more dynamic way, say, to support an ad hoc group of users who get together and wish to collaborate. Creating roles for each such collaboration is not very practical, especially for short-lived collaborations.

Active Spaces are often used in more decentralized ways than conventional distributed systems, and the security administration model has to take account of that. For example, our prototype "smart room", while part of the University and Computer Science department, is most commonly used by members of a few research groups working on a project. The security and access policies are, therefore, agreed upon by these groups. While they must still comply with University and departmental regulations (for example, accounts cannot be created for people without university affiliation), we cannot expect the department administrators to be involved in configuring policy for each of the experiments conducted in this space. Our model is designed to recognize this administrative organization.

While there have been extensions to make RBAC more dynamic and incorporate more information into the access control decision, notably temporal [BBF01] and context [CMA00] information, none of these address collaboration and group permissions which are important for Active Spaces. Our model was designed to support different types of co-operation between users in an Active Space; the objective is to share only the necessary permissions without requiring administrator intervention for most tasks.

One of the distinguishing features of Active Spaces, when compared to traditional computing, is the user-oriented and application-oriented nature of computing. An Active Space can be used for different activities and permissions change based on the activity. Different devices or services may participate in these activities, and permissions need to be controlled.

So the main requirements for an access control model for Active Spaces are:

- Permissions depend on contextual information (in addition to user identity), so the model must be able to represent this information.
- Users bringing their own devices into the space control them; thus devices in a space may belong to, and be administered by, multiple different users.
- Collaboration between users is common, and must be supported easily.
- Decentralized administration of security policies is required, since Active Spaces are usually part of a larger system, and the task of security administration may be split between the larger organization and the component Active Space.

We present our model for Active Space Access Control based on these requirements.

4.2 Model

Our access control model is based upon RBAC, where the basic concept is a role. RBAC simplifies security policy administration by splitting the task into two parts: user-role assignment and role-permission assignment. Users are assigned "roles" when they enter a space, and access control policies for services in the space are created by allocating permissions to these roles. The objects in our model are the services and devices in an Active Space. The possible access rights to a service are the set of operations that can be performed on it. Each service has an access list associated with it, which contains a list of roles and their correponding permissions for this service.

To simplify administration, our system recognizes three kinds of roles: system roles, space roles and application roles. Access control enforcement in an Active Space is performed in terms of space roles; the other two kinds of roles are mapped into space roles. A *system administrator* creates user accounts and assigns them appropriate *system roles*, based on their rights and responsibilities within the system. The *system* here refers to an administrative domain such as a university or company. System roles are assigned permissions to all system resources, based on their task requirements.

We introduce the concept of a *space role*, with associated *space permissions*, which are just permissions to resources within the space. Access control policies for an Active Space are expressed in terms of these space roles and permissions, and this simplifies the task of the space administrator. User accounts are never assigned directly to space roles—the space administrator creates a mapping from system roles to space roles, and when a user with a system role R_{sys} enters an Active Space A, she is automatically assigned a space role R_{space}, which is restricted to a set of permissions that make sense within the space. Figure 4.1 shows how system roles are mapped to space roles.

For example, a space administrator could create a space role of *videoconferencer* that is allowed to setup videoconferences, and decide that only users with a system role of *faculty* could be assigned this space role. While a *faculty* member may have many other permissions in the system, when the role is mapped into a *videoconferencer*, it is only allowed permissions related to that task.

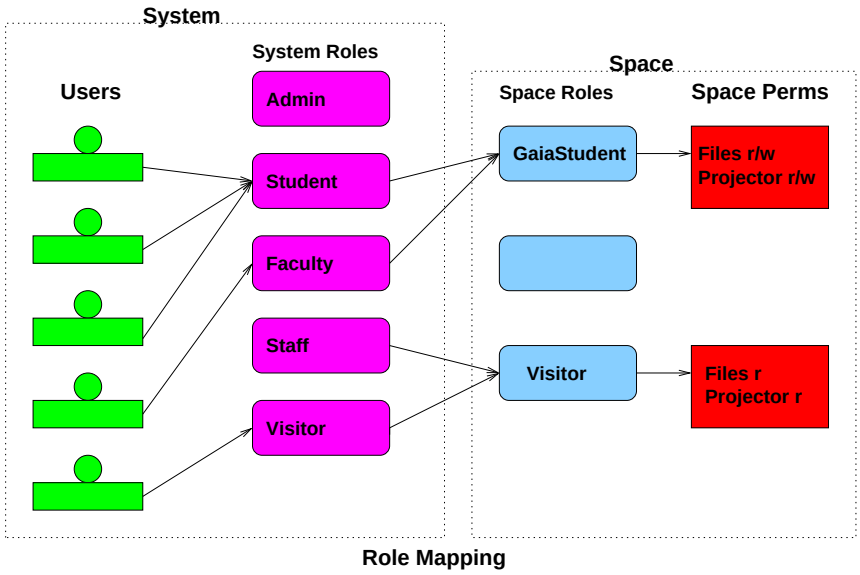


Figure 4.1: System roles mapped to space roles

A user's permissions within a space cannot exceed his or her permissions within the entire system, i.e. the permissions assigned to a space role are always a subset of the permissions assigned to the corresponding system role.

Application roles are used to specify access control policies for applications. These application roles are then mapped into space roles by the space administrator. For example, as shown in Figure 4.2, an application for a seminar may have two application roles—`speaker` and `audience`. The application developer can decide conceptually what the functions of different participants in the application are, and specify this conveniently in terms of application roles—e.g., a speaker must have access to a projection device and audience members must have read access to the slides used. The specific permissions that a `speaker` requires depends on the devices in the space that are running the application. When the space administrator decides that the application can run in a space, he or she also performs the mappings from application roles to space roles. Thus, application roles and system roles are mapped by the space administrator to appropriate space roles, and access control within a space is only enforced in terms of these space roles.

The protection domains in our model are defined by the *mode* of the space. When the space switches modes, new roles may be dynamically created, based on the context of the space, and permissions automatically assigned to them. The security state of the system is represented by the set of access lists associated with the various services in the space.

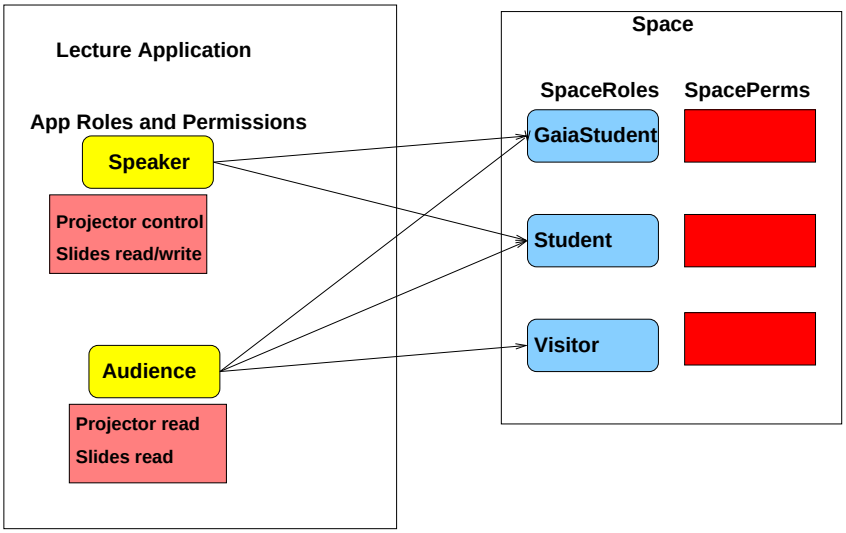


Figure 4.2: Application roles mapped to space roles

4.2.1 Space modes

As users in an Active Space cannot be completed isolated from each other, their permissions affect each other, and the access control model must be able to represent this. We introduce the concept of space *modes* for this purpose. An Active Space is typically used for collaborative applications, by a group of users in the room. There are different kinds of groups that may work together, with different implications for access control. Our model recognizes one mode of individual access and three modes of group access:

The space is in *individual use mode* when only a single user is present. In this mode, this user is allowed access to all the space resources that the access control policies allow her space role.

Group modes, with multiple users in the space, are the most typical usage mode. There are different types of group activities, which involve different modes of space usage.

1. The most common group mode is the *shared* mode, where the Access Control System grants every member of the group the same permissions. In this mode, the users do not have any special trust relationship between them. For such a group, the permissions allowed are only the intersection set of their individual permissions. So a user's set of permissions to the shared resources in a space may decrease if another user (with lower privileges) enters the space, but can never increase. This is the default mode in our system.

2. Another common mode of usage for an Active Space is for a group of users to *collaborate* on a particular application. In this mode, users trust the people they are working with (for this application), and delegate their permissions to the group. The permissions valid in this group are thus the union of the set of permissions of the individual members. Each member of the group has the same set of permissions, but this set could be more than the intersection set.

   While this increase of permissions appears dangerous at first glance, these increased permissions are only valid for the duration of the session, and the space never switches into this mode without explicit on-demand authentication. This mode is only used by a set of users who *want* to cooperate with each other for a particular task. Thus they are able to perform certain activities as a group that they would not be able to do individually. A simple example of this kind of activity is the meeting of the executive committee of an organization—committee members may only have permission to make budget decisions as a committee.

   Thus, the collaborative mode is useful for two kinds of situations: activities by a pre-defined group of individuals (such as the executive committee), or by an *ad hoc* group of users who get together to collaborate on a project. In the former case, permissions have been pre-defined to take advantage of a collaborating group (e.g., no individual space role has sufficient permissions to access the budget, but the set of space roles of president and treasurer *together* do). An example of the latter case is a group of people bringing all their resources (i.e., permissions in this space) to bear upon a problem they are trying to solve.

   The collaborative mode is thus an appropriate model for many uses of Active Spaces.

3. *Supervised usage* is an alternative mode for a group of users in an Active Space, when some users need more permissions than the group to complete an activity. For example, a lecturer might need more permissions than the listening students. We allow a "supervisor" to have more permissions than the default group-mode permissions. Unlike the group modes, where the permissions of all users are set to a common level, in supervisor mode, the supervisor gets more permissions than the group. The supervisor does not, however, obtain more permissions than his original system role is allowed.

   Not all system roles are allowed "supervisor" privileges. When an individual (authorized) user requests supervisor privileges, the space switches to *super-*

Table 4.1: Mode switching in an Active Space

	IND	GROUP		
		SHARED	COLLAB	SUPER
IND	-	*switch*	-	-
SHARED	*switch*	-	*switch*	*switch*
COLLAB	*switch*	*switch*	-	-
SUPER	*switch*	*switch*	*switch*	-

vised mode, a *session* with supervisor privileges is created for this user, and the permissions of other group members remain unchanged from the *shared* group mode.

The space switches automatically from *individual* to a *shared* mode (after notifying the users) when a second user enters it. The collaborative and supervisor modes are only entered if explicitly requested.

The modes of access are similar to protection domains used by traditional operating systems. These group domains are automatically created when a group of users is present in an Active Space, and Table 4.1 shows the allowed domain switches in our system. Traditional operating systems like UNIX usually set a protection domain per user or per process, which is independent of others who may be using the machine. In contrast, in Active Spaces, the mode is dependent on the set of users who are sharing the space, and an additional user entering/leaving can cause the domain to switch.

The access control decision in our system thus depends on the mode of the space, the requester's space role, the access control policy of the space and the operation being attempted.

4.3 Policy specification

Access Control policy development consists of defining methods to control and modify the access lists of services. Access to these services' interfaces in an Active Space is controlled via policies set up by the space administrator. This is the MAC policy for the space.

Users may also bring their own devices, such as personal laptops or PDAs, into an Active Space, and can exercise discretionary access controls over them. However, if users want to access resources belonging to the space, they have to present their credentials with these requests (as usual), and the regular MAC policy will apply to those requests.

We present the policy specification using the guarded command notation, similar to the guarded command language[Dij75; Sch00]. A guarded command is represented as a (guard $\longrightarrow$ command) where a sequence of guards is followed by a sequence of actions. Our policy specification consists of three parts. First we describe the state variables and functions. Next, the access control decision is specified by the Allow method. All other rules specify transitions that change the state of the access lists.

4.3.1 State variables

The state variables that describe our system are:

U : **set of** USERS
R_{sys} : **set of** SYSTEMROLES
R_{space} : **set of** SPACEROLES $R_{sys} \subseteq R_{space}$
R_{grp} : **set of** GROUPROLES $R_{grp} \subseteq R_{space}$
R_{app} : **set of** APPROLES $R_{app} \subseteq R_{space}$
R_{dev} : **set of** DEVICEROLES
S : **set of** SERVICES (objects in the system)
OD : **set of** OWNEDDEVICES $OD \subseteq S$
AL_s : **set of** $\{\langle r : \text{roles}; m : \text{methods}\rangle\}$ (Access list for a service $s \in S$)
A : **set of** all $AL_s : s \in S$ (conceptual Access Matrix)
Mode : **enum** {Ind, Shared, Collab, Super} (space modes)
$\mathcal{C}$: **set of** CREDENTIALS which are one of
$\text{typeof}(u : \text{USERS}; r : \text{SYSTEMROLES})$
$\text{owns}(u : \text{USERS}; o : \text{OBJECT})$
$\text{exports}(s : \text{SERVICES}; m : \text{METHODS})$
URA : **set of** $\{\langle u : \text{USERS}; r : \text{ROLES}\rangle\}$ User-role assignment
AS : Current Active Space; users, services and ALs
CU : **set of** users currently in space AS
RPA : **set of** $\{\langle r : \text{ROLES}; s : \text{SERVICES}; m : \text{METHODS}\rangle\}$ (conceptually the union of all ALs).
CRT : **set of** $\{\langle u : \text{USERS}; r_{sys} : \text{SYSROLES}; r_{space} : \text{SPACEROLES}\rangle\}$
(Current role translation for users in space AS)
SysAdm $\in R_{sys}$, SpaceAdm $\in R_{sys}$, system and space administrator roles

4.3.2 Functions

The **currentrole** function takes a role and returns its current space role, depending on the space mode.

$$\boxed{\textbf{currentrole}(r, mode) : \text{ROLES} \times \text{Mode} \rightarrow \text{SPACEROLES}}$$

The **access control decision** checks credentials that accompany a request for a method of a service, and returns true if the method is allowed. The decision depends on the space mode and the requester's current space role.

$$\boxed{\begin{aligned}\textbf{allow}(u, s, m) \wedge \text{typeof}(u, r) \in \mathcal{C} \wedge\ & (s \in S) \wedge \text{exports}(s, m) \\ & \wedge (\text{currentrole}(r, mode), m) \in AL_s \longrightarrow \textbf{true}\end{aligned}}$$

4.3.3 State transitions

The state transitions are the operations that change the conceptual access matrix A, i.e. by changing the roles or methods in the service access lists.

System policy

The SysAdm role is the only role allowed to add and remove users, roles, devices and services from the *system*.

$$\begin{aligned}&\mathsf{AddUser}(u_{adm}, u) \wedge \text{typeof}(u_{adm}, \mathsf{SysAdm}\ ) \in \mathcal{C} \longrightarrow U := U \cup \{u\} \\ &\mathsf{RemoveUser}(u_{adm}, u) \wedge \text{typeof}(u_{adm}, \mathsf{SysAdm}\ ) \in \mathcal{C} \wedge \{u \in U\} \longrightarrow U := U \setminus \{u\}\end{aligned}$$

Specifications for AddSysRole and RemoveSysRole, AddSysService and RemoveSysService, AddUserToSysRole and RemoveUserFromSysRole are similar: the system administrator credential is checked, and users, roles and user-role assignments are added or deleted. They are omitted here for brevity, and we proceed to the specification of AddPermToSysRole.

$$\begin{aligned}\mathsf{AddPermToSysRole}(u_{adm}, r, s, m) \wedge \text{typeof}(u_{adm}, \mathsf{SysAdm}\ ) \wedge\ & (r \in R_{sys}) \\ & \wedge (s \in S)\ \wedge (m \in M) \wedge \text{exports}(s, m) \longrightarrow \\ & AL_s := AL_s \cup \{\langle r, m\rangle\} \\ \mathsf{RemovePermFromSysRole}(u_{adm}, r, s, m) \wedge \text{typeof}(u_{adm}, \mathsf{SysAdm}\ ) \wedge\ & (r \in R_{sys}) \\ & \wedge (s \in S)\ \wedge (m \in M) \wedge \{\langle r, m\rangle\} \in AL_s \longrightarrow \\ & AL_s := AL_s \setminus \{\langle r, m\rangle\}\end{aligned}$$

The system policy specification similarly specifies all functions for management of the sets U, R_{sys}, URA, S and one AL_s for each service s in S.

Space policy

In each Active Space, a space role is automatically created for each system role. In addition, the space administrator can create additional space roles.

$\forall r \in R_{sys} : R_{space} := R_{space} \cup \{r\}$
$\mathsf{AddSpaceRole}(u_a, r) \wedge \text{typeof}(u_a, \mathsf{SpaceAdm}\ ) \longrightarrow R_{space} := R_{space} \cup \{r\}$
$\mathsf{RemoveSpaceRole}(u_a, r) \wedge \text{typeof}(u_a, \mathsf{SpaceAdm}\ ) \wedge (r \in R_{space}) \longrightarrow R_{space} := R_{space} \setminus \{r\}$

When the first user enters an empty space, the EnterSpace method is called.

$\mathsf{EnterSpace}(u) \wedge CU = \phi \wedge \text{typeof}(u, role) \in \mathcal{C} \longrightarrow$
$CU := CU \cup \{u\}$
$CRT := CRT \cup \{\langle u, role, role\rangle\}$
Mode $:= Ind$

When there are multiple users, the group roles are automatically created from the permissions of the group members.

$\mathsf{CreateSharedRole}(CRT) \longrightarrow$
$\forall AL_s : s \in AS$ (all ALs in space),
if $(\forall u \in CU, (\langle u, role\rangle \in \mathcal{C}) \wedge \{\langle role, m\rangle\} \in AL_s)$
then $AL_s := AL_s \cup \{\langle r_{shared}, m\rangle\}$

$\mathsf{CreateCollabRole}(CRT) \longrightarrow$
$\forall AL_s : s \in AS$ (all ALs in space),
if $(\exists u \in CU, (\langle u, role\rangle \in \mathcal{C}) \wedge \{\langle role, m\rangle\} \in AL_s)$
then $AL_s := AL_s \cup \{\langle r_{collab}, m\rangle\}$

Note that r_{shared} gets permission m iff it is in the intersection set of permissions of each role in CU, while r_{collab} gets all permissions m that are in their union.

When users enter a non-empty space, the following actions are performed.

$\mathsf{EnterSpace}(u) \wedge CU \neq \phi \wedge \text{typeof}(u, r) \in \mathcal{C} \longrightarrow$
$CRT := CRT \cup \langle u, r, r\rangle\}$
$\mathsf{CreateSharedRole}(CRT)$
$\mathsf{CreateCollabRole}(CRT)$
$\mathsf{SwitchToSharedRole}(CRT)$
$CU := CU \cup \{u\}$
$\mathsf{SwitchToSharedRole}(CRT) \longrightarrow$
$\forall u : u \in CU \wedge \text{typeof}(u, role) \in \mathcal{C}$
$CRT := CRT \cup \{\langle u; role; r_{shared}\rangle\}$
Mode $= Shared$

These roles are also re-computed when a user leaves a space.

$\mathsf{LeaveSpace}(u) \wedge u \in CU \wedge \mathrm{typeof}(u, r) \in \mathcal{C} \longrightarrow$
$CU := CU \setminus \{u\}$
$CRT := CRT \setminus \{\langle u, r, r\rangle\}$
$\mathsf{CreateSharedRole}(CRT)$
$\mathsf{CreateCollabRole}(CRT)$
$\mathsf{SwitchToSharedRole}(CRT)$

The CurrentRole method is used to obtain the current space role of a user.
$\mathsf{CurrentRole}(r, \mathrm{Mode}) \wedge \{\langle u, r, r_{space}\rangle\} \in CRT \longrightarrow r_{space}$

Supervisor mode

$\mathsf{GetSuper}(r, s, m) \wedge (Mode = \mathsf{Super}) \wedge \mathrm{typeof}(r, \mathsf{super}) \wedge (\{\langle r, s, m\rangle\} \in AL_s) \longrightarrow$ **true**

Application policy

Policies for an application can be specified in terms of application roles and permissions. The space administrator creates a space role for the application role (if necessary) and maintains a list of system roles that are allowed to switch to this space role. When the set of system roles is updated, this list needs to be updated.

$\mathsf{map}(u_{sp\_adm}, r_a, r_{space}) \wedge \mathrm{typeof}(u_{sp\_adm}, \mathsf{SpaceAdm}\ ) \longrightarrow R_{space} := R_{space} \cup \{r_a\}$

Discretionary access policies

For every "owned" device D that is to be allowed in the system, the system administrator must first create the $\mathrm{DeviceOwner}_D$ system role, and add the owner of the device to this role. The device owner can then set DAC policies for the device.

$\mathsf{AddOwnedDevice}(u_{adm}, owner_d, d) \wedge (u_{adm} \in \mathsf{SysAdm}\ ) \wedge\ (owner_d \in U) \longrightarrow$
$S := S \cup \{d\}$
$OD := OD \cup \{d\}$
$\mathsf{AddSysRole}(u_{adm}, \mathrm{DeviceOwner}_d)$
$\mathsf{AddUserToSysRole}(u_{adm}, owner_d, \mathrm{DeviceOwner}_d)$
$\mathcal{C} := \mathcal{C} \cup \mathrm{owns}(owner_d, d)$
$\mathsf{AddDeviceRole}(u_d, d, r_d) \wedge \mathrm{owns}(u_d, d) \longrightarrow R_{dev} := R_{dev} \cup r_d$
$\mathsf{AddUserToDevRole}(u_d, u, r_d) \wedge \mathrm{owns}(u_d, d) \wedge (u \in U) \longrightarrow URA_d := URA_d \cup u_d$

$$\mathsf{AddPermToDevRole}(u_d, r_d, d, m) \wedge \mathrm{owns}(u_d, d) \wedge (d \in OD) \wedge \mathrm{exports}(d, m) \wedge (r_d \in R_d) \longrightarrow AL_d := AL_d \cup \{\langle r_d, m \rangle\}$$

4.4 Validation

Based on our specification, we give an informal proof sketch about the safety properties of our access control model. The proof assumes that the credentials were generated correctly and the keys of the trust authority were not compromised. Given this assumption, we claim that in the conceptual access matrix A in our system, the existence of an access right of the form $(\{\langle role, method \rangle\} \in AL_s)\ \forall AL_s : s \in S$, is synonymous to the possession of unforgeable $\mathrm{typeof}, \mathrm{owns}$, or $\mathrm{exports}$ credentials that collectively authorize the entry of that right into their respective AL_s. This guarantees that only authorized access to services are allowed in our system.

The proof proceeds by examining all the state transitions and evaluating each state transition rule to guarantee there are no "leaks". A leak occurs when a state transition rule can add an unauthorized entry into an AL. Since we have two types of policies MAC and DAC, we examine the transition rules in turn. For our MAC policy, we observe that whenever we add a new role and method into an access list, we require that the entities (here any user $u \in R_{sys}$) making the request produces a valid typeof credential and a suitable $\mathrm{exports}$. This guarantees that only administrators can add rights to methods that are defined for a given service.

For the DAC policies for user-owned devices, the $\mathrm{owns}(\mathrm{owner_d}, d)$ credential can only be added to the system by administrators. This credential cannot be delegated to anybody other than the owner of the device. A device owner can create roles and add permissions to their own access list, and add users to their private roles. All rights to device interfaces are authorized by the device owner. In addition, by transitive closure of the typeof and owns credentials, these rights are authorized by the system administrators as well, imposing MAC on top of DAC. Therefore, given an initial configuration with only authorized rights, the set of all the reachable states in our conceptual access matrix are also authorized.

Switching domains is governed by the modes of the space, and does not add any new access rights to the matrix, except in the case of the collaborative mode. Collaborative mode sessions are only created when a group of users *who trust each other* want to use the space for a particular activity. The amplification of rights that may occur lasts only for the duration of the session, and are not added into the access matrix for further use.

4.5 Discussion and evaluation

We evaluate the suitability of our model for Active Spaces by discussing its applicability for Active Space applications, and comment on some features and limitations.

4.5.1 Active Space applications

To evaluate the access control model, we considered the dozen or so applications we use commonly in our prototype Active Space. Most applications are started by a user in the space, and other users may sometimes be allowed to modify application behavior. By default, the initiator of the program can terminate the program. Other actions, such as adding or removing inputsensors or presentations (which are analogous to controllers and views in the model-view-controller paradigm), may be restricted, depending on the permissions to the available resources in the space.

We classify the applications into the following categories for the purpose of access control. Some applications may be used in multiple ways and show up in multiple categories.

- Single-user applications: Applications like mp3player and some games can be used when the room is in *Individual* mode. Access control for these applications is straight-forward—typically they use a small number of services, and access to other devices is not required.
- Multi-user/supervised-usage applications: Various applications such as the PPT application for presentations, or the mp3 player can be used in multi-user mode. These applications may sometimes be used in supervised mode if some extra permissions are required. The most common case in which this happens is the use of some equipment in the room, such as a projector or the camera, whose permissions are restricted to professors (and sometimes to administrators).
- Collaborative applications: As mentioned earlier, there are two types of collaborative applications. One type requires the presence of certain users before permissions are activated—we have experimented with this sort of application, for example, by requiring that at least 3 persons were present before starting a presentation. The other mode of collaboration supported is where users present agree to share permissions from roles they activate. We find that users typically do not use this "union of permissions" mode of collaboration

as much as the "supervised" mode—typically there are a few permissions that need sharing for an application, but the supervised mode suffices.

- Space applications: There are various long-running applications that run in the space, *e.g.* the x10 application to control the lights, the cameracontroller application to control the cameras in the room, and the tickertape application that can display messages in the space. These applications behave like system services, and are typically launched with system credentials. These services cannot be disabled by most users. Users may be allowed to interact with these applications, *e.g.* moving or aiming the camera, or sending messages to the tickertape application.

For any of these applications, users can bring their personal devices into the space, and have them participate in these applications. While joining the space, the personal device registers its owner, and, optionally, an access list for services it runs.

The model supports only a limited form of delegation, via the *supervised* mode. Since new permissions from the supervised mode are not added into the access matrix, this simplifies the security analysis. However, we recognize that the lack of delegation may prove a limitation for certain applications.

This model appears to be expressive enough to satisfy our current Active Space applications.

4.5.2 Comparison with existing models

Since Active Spaces are still being used in experimental ways, flexibility about the kinds of access control policies that can be supported is important. Our model is a form of RBAC, and supports DAC and MAC security policies. We discuss here the differences from traditional RBAC.

Access control policies for Active Spaces typically allow a role (or set of roles) permissions to perform certain actions under certain contextual conditions. The RBAC has a set of users, roles and permissions. Role activation decides which permissions are active in a session. Roles are fixed sets of permissions, which are suitable for reasonably static task descriptions.

To cope with the dynamism of Active Spaces, our model supports more dynamic roles—system roles and application roles get mapped into space roles at run-time, and access control enforcement takes place in terms of space roles. Since rights amplification cannot occur in this mapping, the system and application roles are purely for administrative convenience, and do not affect the safety considerations.

Group roles are also created dynamically, based on the active permissions in the session. Dynamically creating roles allows the set of permissions to be more adaptive to the current state of the space (such as the users currently in the space) without the administrative overhead of pre-configuring policies for all possible situations.

4.5.3 Collaboration support

Active Spaces need to support a range of collaborative activities between users who trust each other to varying extents. Our model handles this by recognizing different modes of collaboration, in which different sets of permissions are shared.

The shared mode enables the intersection set of permissions of the users present; this is safe, but possibly too restrictive. It is a useful mode for sharing the space between mutually non-trusting users, especially when the space cannot definitively recognize which of the users present is performing some action (such as entering data via a shared touchscreen display).

Supervised usage represents a form of limited delegation—a user in the room may be permitted certain actions during a particular session *in the presence of* a supervisor. This maps well to real-world situations where students are often allowed to use a classroom projector after the professor has unlocked it. In our model, the permissions do not last beyond the presence of the supervisor.

Collaborative mode allows some rights-amplification, however, it is rarely used. We argue that explicitly providing support for this mode of operation is useful, since users otherwise have to find ways around the model to perform activities that require this support.

We posit that these modes are sufficient to represent all possible modes of collaboration that occur. Formal study of the completeness of these collaborative modes is left for future work.

4.6 Conclusion

In this chapter, we provide the formal specification for our access control model. The main differences from RBAC are the existence of three types of roles, and the various space modes to represent different modes of collaboration. This model is expressive enough for the requirements of Active Spaces. It is flexible enough to support a variety of access control policies, such as DAC and MAC, and supports the

various modes of collaboration in a secure manner. A limited form of delegation is supported via the “supervised usage” mode.

5 System Design and Implementation

In this chapter, we describe the architecture and implementation of the Gaia Access Control system. Section 5.1 outlines the design objectives. Section 5.2 describes the system architecture, and Section 5.3 the implementation of the Gaia Access Control System. Section 5.4 presents a security analysis of the implementation, and Section 5.5 a performance evaluation, before the conclusion in Section 5.6.

5.1 Design objectives

Active Spaces pose special challenges for access control because of their context-dependent nature, the variety of mobile and heterogeneous devices and users, and new modes of interactions with the sytem. From our model, we obtain the following design goals for an access control system implementation:

- User permissions in Active Spaces depend not only on the identity of users and the objects being accessed, but vary with system *context*. The Access Control System, therefore, must have access to information about the system context, and should be able to incorporate this information into the access check.
- Users can bring their own computing devices (such as laptops, PDAs or cellphones) into an Active Space and have them become part of the space. In these situations, the owners of the devices should be able to control access to them, which requires supporting *discretionary* access controls (DAC). *Mandatory* access controls (MAC) are also required, to specify policies for an Active Space, especially for devices that are shared by users in the space.
- Most usage of Active Spaces is *collaborative* in nature; such spaces are typically used by groups of users for various applications. The access control mechanisms should facilitate collaboration, by allowing groups of users to interact in a secure manner without requiring administrative intervention.
- The potentially large number of devices and services in the system require a *scalable* approach. Poorly-implemented security mechanisms can cause a

great performance hit, leading to users disabling them in practice. Hence performance considerations are critical in the design of practical security systems.

- The complex environment can also make security *administration* a problem. Manageability of the access control system is an important design criterion.
- *Usability* is important for a practical security system, since accidental misuse of security mechanisms are often responsible for security breaches. Usability has many aspects; we consider both end-user and administrative usability to be important for our system.

In the following sections, we describe the architecture and implementation of the access control system and how it addresses these objectives.

5.2 Architecture

We now present the architectural design of the Gaia Access Control System, first providing an overview of the access control system before describing the components of the system in more detail. Broadly, the access control system works as follows:

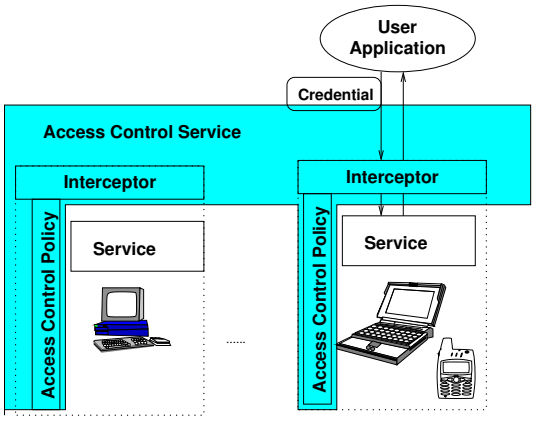


Figure 5.1: Gaia Access Control System architecture

- The space administrator configures access control policies for all devices and services in the space. All access to devices is through services, and these services are associated with the appropriate access control policies for the device.
- Users enter the space after authenticating themselves. Physical access to the space is restricted to legitimate users. The space knows who its current users

are. Anonymous use may be allowed by the space policies, but the space will still know that an anonymous user is present.

- Users run applications in the space, which make requests to services on their behalf. The access control system intercepts all requests to services, checks the accompanying credentials, and enforces the policies associated with the services.

The basic architecture of the Gaia Access Control System is shown in Figure 5.1.

For the access control system to be able to restrict access to authorized users, it must be able to identify the users. We describe this process next.

5.2.1 Authentication

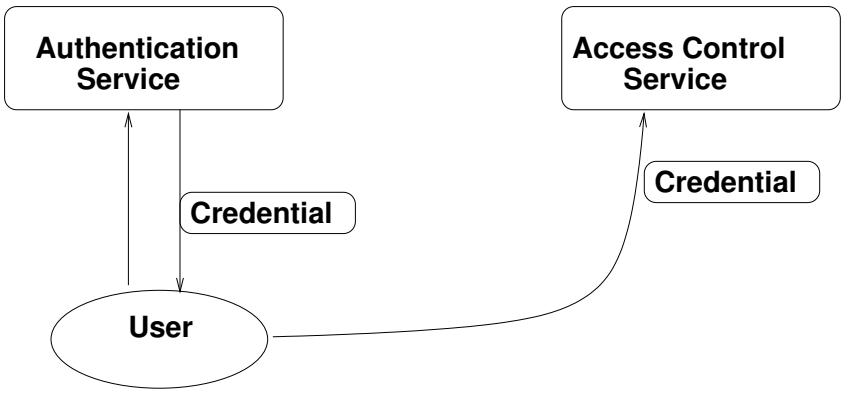


Figure 5.2: Authentication and access control

Traditional access control schemes typically base authorization decisions on the identity of the requester, so recognizing users accurately is an important requirement for an access control system. In Active Spaces, a variety of other factors, such as the system context, may also affect user permissions, but user identification is still required. User identification is not part of this research; we rely on existing services such as the Gaia Authentication Service to perform this function. The architecture is shown in Figure 5.2. However, we discuss here the practical challenges in using these or traditional authentication systems in our environment.

In principle, any mechanism to identify users and issue them appropriate credentials that they can use to make requests within the Gaia system will serve our purpose. The Access Control System also needs to have information about which users are in the space, since permissions may depend on this. Precise identification of the user may not be essential in certain contexts—a user may choose to be "anonymous" in the space, and use only such privileges as the space allows anonymous

users. However, even here, the system needs to know that *someone* is in the space, as activities permitted in an empty space are probably different from those allowed in a space containing an unidentified user. We assume that any user physically entering the space is noticed, either by the Authentication system which might challenge him to authenticate himself, or by some other sensing device like a camera or a doorlock server. This is a reasonable model for Active Spaces such as smart rooms with controlled physical access. For Active Spaces that have more uncontrolled access (such as a shopping mall), it may be more accurate to assume that there can always be unidentified users around, and configure access control accordingly. Whatever authentication mechanism we use, the Access Control System needs to know how many users are in the space. It may also have more information about their identity or location, as provided by the Gaia Authentication Service or Gaia Location Service. If available, this information may also be used for access control.

After users have been identified, they can be issued with credentials certifying their identity. In theory, they can then use these credentials to obtain services from the system. However, there are several practical challenges in designing such a system. We describe two techniques that have been used to assign credentials to users in the Gaia system, and point out the issues related to using them for access control.

Identity-based credentials

The Gaia Authentication Service deals with user identification by having users "login" to the Gaia system. It uses a variety of techniques to identify users, such as passwords, various types of biometrics (fingerprint and iris scanners), and tracking devices such as RFID badges. Once identified, the user is issued with a credential. This credential can be used to make requests within the Active Space. In the Gaia system, users have permissions within an Active Space based on their identity and their current *space role*, which depends on the current activity within the space.

While passwords are the most commonly-used mechanism for user authentication, there are well-known problems with them [Kle90]—"good" (i.e. not easy-to-guess) passwords are often hard for humans to come up with or remember. In Active Spaces, where users may not even be using a keyboard, limitations of passwords are even more apparent. Biometrics may prove to be a useful mechanism for user identification, but currently available technology has limitations. We have experimented with biometric devices such as fingerprint readers and iris scanners in our Active Space, but they are still fairly intrusive and slow. Recognizing a fingerprint can take upto a few seconds, and often requires repeated attempts. This

is not a feasible option for frequent re-authentication. Less intrusive methods like face recognition by tracking cameras sound attractive, but are not reliable enough yet. Different methods of authenticating users have differing levels of accuracy and usability. The Active Space Authentication System uses a combination of these mechanisms to identify users with varying levels of confidence, depending on the methods used [AMRCM03].

One problem with credential-based authorization in Active Space environments is credential management by the user. After the user is issued a credential, it needs to be stored securely, presented while requesting services, and associated by the system with the appropriate user.

To understand this problem, consider the case of a user, John, who moves around and uses various devices in an Active Space. John authenticates himself to the Gaia system by means of a fingerprint scanner. The Gaia Authentication Service is satisfied that John is in the Active Space, and is standing near the fingerprint scanner. If John is the only user in the space, it may be appropriate to assume that any request made within the space was issued by John. However, if there are other users present in the space, how can the space distinguish requests from the different users? Since John was just at the fingerprint scanner, it may be appropriate to assume that any commands issued from that part of the space within a certain time interval were issued by John.

However, if John then proceeds to walk up to the plasma display wall and use the touchscreen to enter commands to interact with the space, there is no way for Gaia to realize that these commands were issued by John, rather than any other user in the space. John could re-authenticate himself at another identification device near the display wall, in effect informing the space of his new location, and proceed from there. However, this repeated re-authentication is tedious if it requires John's intervention. It may be usable if the space could track his location as he moves around, without his having to repeatedly use fingerprint scanners or enter passwords.

For the system to know for certain that a request comes from John, the request should be accompanied by his credential. This raises the problem of credential storage—when John identifies himself to the Authentication Service by means of a fingerprint scanner, where is the Authentication Service to send the credential to? One possible solution is for John to have some personal storage on a device such as a PDA or a cellphone. Upon identification, the Authentication Service can then send the credential to "John's personal device". This device then behaves as John's proxy—he can use this device to launch applications in the space, and the

requests will be accompanied by his credential. Since applications can be dynamically moved to different devices in the space, this may be a reasonable approach for certain users and applications.

However, the scheme described above fails if John does not have a personal device with him. In that case, another option is for the Active Space to provide a credential storage service. However, this still requires John to identify himself to this service whenever he wants to use his credential, say, to launch an application in the Active Space. While this may be suitable for certain low-frequency operations, it is too cumbersome for general use with all Gaia activities.

From the above example, we identify the following three criteria:

1. Users in an Active Space are not limited to using a single device. It must be possible to have them authenticate themselves once and then be allowed to use all the devices they are authorized to. Repeated re-authentication as they move around the space is not a feasible option.

2. Identifying user location within an Active Space is useful for access control. Having Gaia do this automatically without requiring repeated user interaction is desirable.

3. It is useful for users in Active Spaces to have at least a small amount of local secure storage, to store credentials.

We explore this idea further with the use of location-tracking tags, which can be used to associate users with credentials in Gaia.

Location-based credentials

In some situations, the user's identity may not be as important for allowing access as knowing that he or she satisfies the criteria required to obtain a particular service from the system. For example, any user within a building may be allowed access to information about activities occurring within the building. So if a user can prove she is within the building, she is allowed access to the relevant services. With the help of location-tracking systems, it is possible to perform *location-based access control*.

In an ideal world, tracking cameras with face recognition capabilities would be able to identify users exactly as they walked around the room, and use that identification for access control. The current state of the technology is still some distance from that ideal—face recognition is nowhere near accurate enough to be relied on for distinctly identifying users in the space. We get around this problem by having

the user co-operate with the tracking system by carrying a badge, so that we do not have to rely on automatic visual recognition of the user. The Gaia Location Service can use any of a variety of sensing techniques, such as infra-red (IR) beacons or radio-frequency (RFID) badges, to provide location information about users.

Our prototype Active Space is equipped with UbiSense [WWB03] location-tracking equipment, which is capable of locating UbiSense tags to an accuracy of 6 inches in 3D. The software platform uses sensor data from the tags to update a model of the space that changes in real-time. Users are issued tags when they enter the space, and these tags identify them within the space. With this scheme, a user enters the space, identifies himself (to the Authentication System) and then picks up a tag. For the rest of the session (until he logs out), the tag is associated with this user. Permissions associated with this user follow the tag around the space, and requests for services are attributed to the user whose tag is located near the request source. Thus, the tag serves as a physical credential for the user, and solves the problem of associating credentials with a user as he moves around the Active Space. The actual credentials are maintained with either the Authentication Service or the Access Control Service, and the tag serves as a pointer to the correct credential for each user in the space.

One drawback of this scheme are that user permissions follow the tag, so someone stealing the tag could steal permissions. However, we expect that a user who loses a tag will notice it (since she will no longer be able to do anything in the space), so this is not a serious threat in practice. Tags cannot be stolen and re-used secretly later, since the mapping of the tag ID to a particular user is only valid until the user logs out. So far, this seems a reasonable approach to the problem.

The key points of this scheme are:

- A user can be identified once on entering the space, and her permissions will be associated with her as she moves around the space. No re-authentication is required for each device she wishes to use.
- Losing or stealing a tag could lead to permissions being lost, but since the association is only for the duration of the session, and the loss of the tag is immediately obvious, this is not likely to be a problem in practice.
- An advantage of this scheme is that it allows users in the space to perform actions in an authorized manner without requiring them to carry a device such as a PDA with them.

5.2.2 Policy management

We use a form of role-based access control (RBAC) to administer security policies within an Active Space. Security administration in this access control system is distributed between the system administrator and the space administrator. The system administrator adds new users and gives them permissions within the entire system, whereas the space administrator is concerned with permissions to the specific devices and resources within an Active Space. This is an accurate representation of administration in most real-world environments, with the delegation of administrating individual spaces to local administrators.

The system administrator's task consists of assigning new users to the system to appropriate roles. The space administrator maps these system roles to appropriate space roles, based on tasks that may be performed in the Active Space. When a new application is to be installed in the space, the space administrator can map the application roles to appropriate space roles, creating new space roles where necessary. The space administrator ensures that no rights amplification can occur during this role mapping. The administration model is designed to minimize administrative actions required at run-time—collaborative groups are automatically formed from the roles of the users present, and do not need special roles to be created.

This division of labor is similar to what happens in today's networked environments—user accounts are created, and users are assigned to "groups" in UNIX or Microsoft Windows systems. Less frequently, new "roles" may be created, when new types of users enter the system. For example, all students in a university may be given a university computer account, but only students taking a certain class may be allowed to use a particular department computer. The departmental administrators are usually separate from the campus system administrator, but will check that the requester is a bona fide student before creating an account on the departmental computers. In our system, the space administrator's task is simplified by assigning space permissions to roles, since roles can be thought of in terms of tasks that users want to complete, and form a natural grouping of permissions that need to be assigned or removed.

5.2.3 Sessions

Sessions in an Active Space represent a configuration of the access control policy. The protection state of the system is invariant for the duration of a session. A session contains the current mapping of users to roles, and the mode of operation of the space. The access control policy for each service is a boolean expression

that contains propositions representing user credentials (and identity or role) as well as those representing system context (such as activity). Since access control is on the critical path, the performance of the actual access check is important. To speed up the permission check, we maintain the policy in the form of a table, such that the run-time cost is just a simple table lookup in a reasonably small table. For each session, this policy is translated and stored in the form of an *access list*, containing an entry for each active role within the space and listing the permissions that members of this role have for this service. The access control decision is thus reduced to looking up the requester's role in this access list, and verifying that it is allowed to perform the operation it is attempting. Since this is a simple table lookup, it can be performed efficiently. The access control decision is based upon the current space role of a user, and the permissions assigned to this role in the access list of the service.

A new session is created when a user enters or leaves a space, or when an application is started. Creation of a new session consists of updating the role mappings in the space, and setting the space mode to a new value if necessary. Access lists for the services may need to be re-computed when the space switches modes.

The space administrator maps application roles to space roles (this has to be done once per application, when the application is installed in the space). On starting an application session, users in the space get assigned to the space roles corresponding to their roles in the application. Sessions are destroyed when applications terminate, or when a new session starts, perhaps by a different set of users gathering in the space. A session is valid as long as the role mappings do not change.

The trade-off with the session approach is that we incur the cost of updating access lists when a session starts; however, this allows us to have a low lookup cost. This trade-off is reasonable if sessions are at least a few seconds long. In practice, sessions in Active Spaces are typically minutes long, as they last the duration of applications. Users entering and leaving may sometimes cause session reconfiguration, and too many users entering and leaving in a short period may cause undue overhead. However, situations like students entering a class will not require reconfiguration, so we expect this scheme to deal satisfactorily with most normal flows of users.

Now that we have described the prerequisites such as authentication and the process of configuring the access control policy, we move on to describe the access control system.

5.2.4 System components

To implement access controls, all attempts to access Active Space resources must be intercepted and checked against the policy before being allowed to proceed. We describe the design of the two components of the run-time system: one that performs interception and the other that checks for authorization.

Request interception

One of the requirements for access control is *complete mediation*, i.e. that all requests are checked by the ACS before being allowed to proceed. This is most commonly achieved by using a *reference monitor*, which intercepts all relevant requests and performs access checks. Every request must be accompanied by a credential which contains sufficient information to authorize the request. This credential (and other relevant information, if any) must be presented to the Access Control Server, which makes the decision to allow the request to proceed or not.

One possible technique to implement this is to have each service interface contain a credential as an additional request parameter. The server first extracts the credential and passes it on to the ACS for permission-checking. Only on receiving authorization from the ACS does the request proceed. However, this approach is intrusive: it requires every service/application developer to be aware of and implement this interface. Client applications have to explicitly manage the user credentials, and servers will each need to implement the functionality of interacting with the ACS. It is difficult to ensure that all services do perform the access control step correctly, since it is implemented as part of the service.

A cleaner mechanism is provided by the CORBA Portable Interceptors [OMG], which allow all requests in a CORBA system to be intercepted, at either client or server side (or both). This interception mechanism can be provided as part of the Active Space system infrastructure. Client-side interceptors can be used to attach credentials to requests, and server-side interceptors can extract these credentials and perform an access check. All this is entirely transparent to the application, which does not need to be modified in any way. More details about the implementation are provided in Section 5.3.

Providing this service as a part of the Active Space has two advantages: it ensures that all services running in the space will have access controls and it relieves application developers from having to be aware of the security interfaces. Installing a service in an Active Space requires the specification of a policy with the ACS (default space policies could be provided and used), and this policy will be enforced

automatically by the Access Control Service in the space. At the client-side, a similar interception mechanism can be used to attach credentials to outgoing requests automatically.

The basic functionality provided by the interception component of the access control system is:

- At the client-side, attach credentials of the calling user. This assumes that the user authentication process in the space provides the credentials.
- At the server-side, extract and check these credentials.

Both these happen transparently to the application, so application developers need not be concerned with implementing security mechanisms. The only visible effect of the ACS is that unauthorized requests fail.

Access control server

The function of the Access Control Server is to maintain the current access policy for the Active Space and to perform access control checks for all requests to services within the space. This leads to the following design considerations:

Availability: Every request to a Gaia service must be checked by the ACS before being allowed to proceed (if authorized). Thus, the Access Control Server must be on-line and available throughout system operation.

The Access Control Server must be initiated at system startup, before any other services are started, since access control is only effective if it implements "complete mediation", i.e., it should not be possible for service requests to bypass the Access Control Service in any way. This requires high-availability from the ACS, since the ACS being unavailable brings the entire system to a halt. This requirement led us to keep the ACS implementation simple. To minimize unavailability due to ACS failure, we allow the ACS to be restarted without requiring a full Gaia reboot. This is enabled by having the ACS use soft-state, so that it can be reconstructed on restart.

Performance: Since every request in the request passes through the ACS, the response time of the ACS is of great importance. The ACS must not be a bottleneck in the system. To achieve this objective, we process the policy and store it in such a manner as to allow for the run-time check to be as efficient as possible.

The system access control policy is expressed as a boolean formula, with the

proposition values depending on the user credentials and the current state of the system at the time of attempted access. To improve performance, this formula is interpreted and reduced to an AL, which is a list of authorized users (or *roles*) and the methods they are allowed to call on a particular service. Thus, at run-time, the access check involves a simple lookup in the AL. Whenever the policy changes, usually as a result of a change in the system context or due to administrator intervention, the formula is re-translated into an AL and loaded into the ACS. We are thus assuming that the rate at which the policy needs to be updated is low enough (typically of the order of once every few minutes) that this works well; this is a reasonable assumption given the environment.

Policy support and dynamic reconfigurability: An important requirement for the ACS is the ability to enforce the variety of policy requirements of the different applications that may use an Active Space. Support for both MAC and DAC policies are necessary, due to the different administrative authority over various devices in the space. Users who bring personal devices into the space will need to retain the ability to control access to their devices, whereas the administrator of the Active Space will set the policy for devices that belong to the space.

Given the high-availability requirement and the dynamic nature of Active Spaces, it is not feasible to restart the ACS whenever a policy change is required. Thus, the ACS must support dynamic reconfiguration of the policy.

Usability and manageability: While mechanisms for fine-grained access control are feasible from an implementation point of view, the configuration complexity can rapidly overwhelm a human administrator. If it is not possible in practice to specify a desired security policy, it is impossible for any system to enforce it. One of the design criteria to help configuration is to require that it be possible for an administrator to obtain the current state of the system from the running ACS. This information may be used by other tools to help with policy configuration, or for system monitoring.

We now proceed to describe the implementation of this system.

5.3 Implementation

The Gaia system is implemented as a middleware "meta-OS" using CORBA. The current implementation uses Orbacus [Orb] as the CORBA implementation for Mi-

crosoft Windows and Linux, and also uses a smaller custom ORB [UIC] for handheld devices. All system services are CORBA services. We, therefore, implement the Access Control Server as a CORBA service. The two major components are an Access Control Server and a Request Interception library, and we describe each of them in detail in the following sections.

5.3.1 Request interception

In the Gaia environment, where system services are implemented using CORBA, the natural choice for intercepting requests and enforcing access control policies is to use CORBA Portable Interceptors[OMG]. Request Interceptors allow one to write and attach portable ORB (object request broker) hooks that intercept all ORB-mediated communication. These hooks can be completely transparent to the service implementation, so service and application developers do not have to worry about security, and the administrator can install the hooks to implement the space access control policy when a service or application is installed in an Active Space.

There are two types of Portable Interceptors—*IOR interceptors* and *request interceptors*. We use request interceptors, which are further divided into *client request interceptors* and *server request interceptors*. These are designed to intercept the flow of a request/reply sequence through the ORB at specific points on clients and servers. The interceptors are installed into the ORB via an IDL interface defined by the CORBA Portable Interceptor specification. There are ten different interception hook methods that can be called at different points in the chain, but we are mainly concerned with:

- send_request(), called when a client sends a request, before marshaling.
- receive_request(), called on the server after the request is demarshaled.

At run-time, the interceptors can examine the state of the request that it is associated with, and perform certain actions. They can access information in the request, insert and extract piggybacked information from a request's service context, redirect requests and/or throw exceptions. Interceptors cannot, however, modify the request parameters or return values.

The interception mechanism is implemented as two libraries: one each for the server-side and client-side interceptors. These are implemented as subclasses of the two CORBA-specified classes, PortableInterceptor::ServerRequestInterceptor and PortableInterceptor::ClientRequestInterceptor. These classes specify the hook methods, such as receive_request() and send_request(), which are overridden to provide the required functionality.

The server-side library intercepts every request going through the ORB before the target server object receives it. The receive_request function implemented in the interceptor expects a user credential to be available in the request service context. It then calls the isAllowed() method of the ACS, providing it with the user credential and the requested target object and method. If access is allowed, the receive_request function completes, and the request proceeds to the target server. If the request fails, the ACS throws a CORBA NO_PERMISSION exception, which is provided to the client application to be handled appropriately. It is, thus, impossible for an unauthorized request to reach a Gaia service in the space.

The main task of the client-side request interceptor is to attach credentials to the requests. Ideally, this process is completely automated, and requires no user intervention. However, some protocols and/or user-intervention may be required to have the library access the credential in a secure manner. We have two implementations of this client library—one using identity-based credentials, and the other using host/location-based credentials. For the identity-based credentials, we assume that the user has authenticated himself or herself to the space, and obtained a suitable credential, which is maintained in a location accessible to the library. For example, we use an environment variable pointing to the location of the credential. Obviously, other users on a multi-user machine should not have access to these credentials.

Our second implementation of client-side interceptors also attaches the hostname of the sending machine to each request. Combined with the use of location information obtained from the various location-tracking mechanisms in the space, this can be used to enforce location-based access control.

Client-side interception provides a simple mechanism to intercept all requests and automatically attach the required credentials, so that requests can present the appropriate credentials when they reach the server.

All applications in Gaia are component-based. A ComponentContainer component runs on each Gaia execution node. Gaia application functionality is implemented in the form of shared libraries (DLLs) that are loaded by the ComponentContainer. The ComponentContainer also performs the CORBA initialization, while loading the ORBACUSExporter library. The interceptor libraries are loaded before initializing the orb. All requests are automatically and transparently intercepted.

Gaia applications developed using the Ubiquitous Application Framework [RC03] consist of a set of communicating components. A Model provides the application functionality, and is controlled by user commands via InputSensors. Output is displayed via Presentations. Controlling access to an application functionality, there-

fore, involves restricting access to the functional methods exported by the model for the application. User operations are typically performed by the InputSensor, which act as *clients* to the Model which provides the application *services*. Thus the InputSensor components need client-side interceptors and the Models need server-side interceptors.

5.3.2 Access control server

The Access Control Server is implemented as a single CORBA service. It is started during the bootstrap of the Gaia system. At startup, it is initialized with access control policies for the various system services. All requests for services in the Active Space cause a query to the Access Control Server, which responds after checking the Access List for the relevant service. During regular operation, users entering and leaving the space may cause mode change and access list reconfiguration. Sessions are also started when applications start and end. Collaborative and supervised sessions are explicitly started, and do not happen automatically.

The interface provided by the ACS has the following important functions:

- isAllowed() is the most-used method. It is called for every request, and uses the information provided to check whether that particular request is authorized. This function is called often and is on the critical path for every request, so performance considerations are important. This function takes three arguments: a user credential, the target service and the target method.
- EnterSpace() and LeaveSpace() are used to inform the ACS about users entering and leaving the space. This information can be provided by the Authentication Service. These functions trigger the reconfiguration of the space modes based on the number of occupants.
- readAList() is used to configure the ACS with an access list for a particular Gaia service. The arguments are the service name, and a filename pointing to the access list.
- dumpSpace() is a diagnostic function to display the access control state of the system.

A complete interface specification for the ACS is presented in Appendix A.

5.4 Security analysis

In this section, we present an informal security analysis of our implementation. We discuss the practical attacks that can be mounted against an access control system and discuss the vulnerability of our implementation to them. While the prototype implementation is not resilient to all the attacks, it is fairly straightforward to implement the defenses for a production system.

The primary task of the access control system is to ensure that only authorized requests are allowed to proceed. This imposes two requirements on the system: being able to intercept every request and attribute it to the correct source (i.e., being able to identify the requester), and being able to decide, based on the system policy, whether this request is permitted. This identifies the following ways in which to attack such a system:

Request Identification: Correct authorization of requests depends on the ability to identify the source of the request accurately. We depend on the Gaia Authentication Service to perform this task. The Authentication Service identifies users and issues credentials. Our concern (for access control) is with the possibility of generating false credentials or stealing or replaying legitimate credentials. Having credentials cryptographically signed by the issuer reduces the likelihood of fake credentials. The use of stolen credentials can be detected and prevented by having users re-authenticate themselves to refresh credentials periodically. We use this mechanism in Gaia, but there is a tradeoff between the intrusiveness of repeated re-authentication and the vulnerability window during which a stolen credential may be used.

Policy Configuration: Misconfiguration of security policy is an obvious vulnerability in security systems. We have addressed some usability concerns to simplify the task of the administrator, but better configuration and monitoring tools would be helpful.

Bypassing access controls: Another problem is if requests can somehow bypass the Access Control Service. Using the CORBA portable interceptors is a simple way to automatically intercept all requests to a CORBA service, and works very well for Gaia, since system services and applications are implemented as CORBA services. Thus, it is not possible to bypass the access control mechanism while requesting Gaia services.

Availability: Since the Access Control Service intercepts all requests, it could be used to try and mount a denial-of-service (DoS) attack on the space. Flood-

ing the Access Control Service with malformed or unauthorized requests can slow down response to legitimate requests. However, the Gaia Access Control System is less vulnerable to DoS attacks than a general Internet service, because most legitimate requests for services in an Active Space come from local devices, and misbehaving or flooding clients can be easily identified and blocked. While we have not implemented this in our prototype, throttling requests from misbehaving clients is feasible.

In our current implementation, the access control service is a single point of failure in the system. While bringing down the access control service makes the entire system unavailable, our architecture uses soft-state for the ACS, which allows it to be restarted during system operation without much effect on clients.

Our current implementation has a monolithic access control service, but for larger or busier systems, it can be implemented in a distributed fashion for improved performance.

Unauthorized devices: Unauthorized devices that never register with the space do not have access controls. This poses a risk for privacy policies, for example, a space policy may require that all recording devices be turned off, but a "stealth" recorder brought in by a user may go undetected.

5.5 Performance evaluation

Access control is on the critical path for all requests in the system, so performance considerations are paramount. The access control system design and implementation were strongly influenced by performance considerations. We have evaluated the implementation of the Access Control System, and present the results here. The evaluation is in two parts: a qualitative evaluation first describes the behavior of the ACS with the help of an example scenario, describing the security state of the system as various operations occur, and then we present results from micro-benchmarks designed to evaluate the scalability of the system with respect to the number of users, devices and contexts.

5.5.1 Qualitative evaluation

In this section, we describe the operation of the Access Control System by working through a representative scenario of how the Active Space is used, and showing the security state of the system at each step.

Space mode: *Empty*

Table 5.1: Security state of Space in Empty mode

Access lists	
Roles	mp3player
RoomUser	start, stop previous, next setVolume, getVolume toggleVisualization storeCurrentTime, getStoredTime
Visitor	stop
Admin	stop

Current Role Translation		
user	System Role	Space Role
Alice	CSstudent	RoomUser

Space mode: *Individual*

Table 5.2: Individual mode session for mp3player

Our "Smart Room" is part of a Computer Science department, and has been operational for over a year. The room contains a variety of computers and displays, and other devices like cameras and location-tracking devices. It runs the Gaia software to manage all these devices and provide a unified computing environment. The room is used for various research and academic activies—seminars and classes are conducted there, students work to develop and test their software, and visitors often stop by to watch demonstrations of the research. We describe a series of activities that are conducted there, and which, between them, represent the various modes of operation. For simplicity, we describe a subset of the devices and how their access permissions change.

The Active Space starts out in Empty mode, with the Gaia services running, but no users present. The Access Control Service is started as part of the Gaia boot, and installs the configured access lists for the system services. No applications are running at this time. The security state of the system is shown in Table 5.1.

A user, Alice, enters the room. When she authenticates herself to Gaia, she acquires the system role of CSstudent, which is currently mapped into the space role of RoomUser and the room switches into *Individual* mode. Alice wants to use the mp3player application to listen to music in the room. When she starts the application, a new session is created. Her RoomUser role is allowed to use the mp3player application, as shown in Table 5.2.

Another user, Bob, enters the room. The room automatically switches into *Shared* mode, with the permissions as shown in Table 5.3. Bob's system role is student,

which maps into a space role of Visitor. Since the Visitor role is not allowed to use mp3player and the *Shared* mode uses the intersection set of the permissions, the next song cannot be played.

Access lists	
Roles	mp3player
Group	stop

Current Role Translation		
user	System Role	Space Role
Alice	CSstudent	group
Bob	student	group

Space mode: *Shared*

Table 5.3: Shared mode session

It is now time for a class, and the professor, Carol, enters the room. The class application session is started, with Carol assigned to the professor role, and Alice and Bob to student space roles. Carol starts the PPT application, which can display Microsoft PowerPoint slides across various displays in the room. She is assigned to the supervisory speaker role, while all others in the room are assigned to the space role of audience. Only the speaker can control the presentation, but all others have access to the content. While the speaker is present, write access to the whiteboard is enabled.

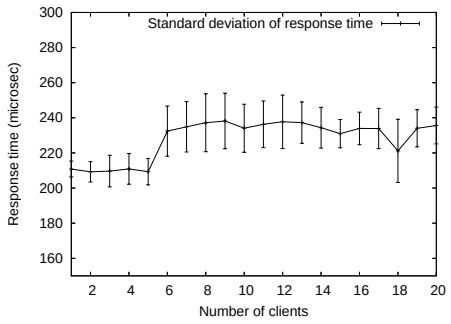
Access lists	
Roles	PPT
Speaker	start, stop previous, next

Current Role Translation		
user	System Role	Space Role
Alice	CSstudent	audience
Bob	student	audience
Carol	professor	speaker

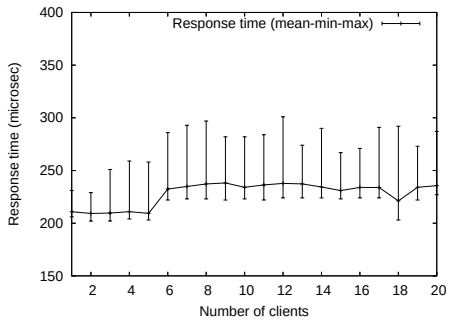
Space mode: *Supervised*

Table 5.4: Supervised session

We finally describe the collaborative mode, when a faculty committee meeting occurs. The meeting application has an agenda that is displayed using the PPT application, and a log that contains the minutes of the meeting. The agenda document is displayed when the meeting application starts, but write access to the log is only permitted if at least two faculty members have entered.



(a) Response time of ACS (mean and standard deviation)



(b) Response time of ACS (mean/min/max over 1000 requests)

Figure 5.3: ACS response time with varying with number of clients

5.5.2 Scalability evaluation

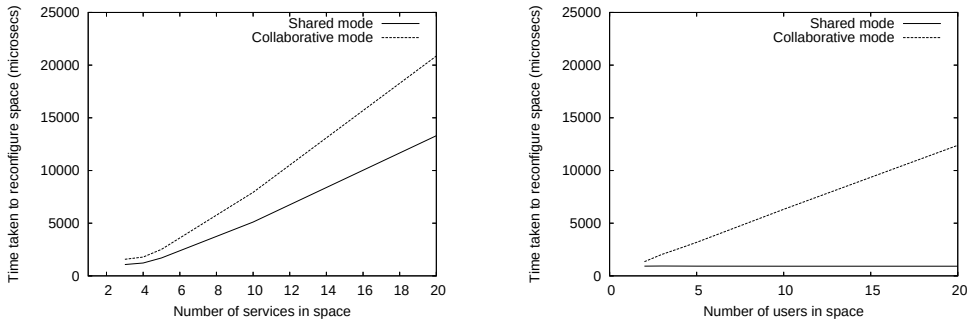
The main performance criterion is that the access control system should not be a bottleneck in the system operation. This has been qualitatively verified by having the access control system operational in our prototype Gaia Active Space for over a year, with negligible performance overheads. In this section, we evaluate the system by means of micro-benchmarks to demonstrate how the Access Control Service responds to increasing load. We measure the response time of the access control server as the number of clients, requests and services change.

All tests are conducted in our prototype Active Space environment. The ACS runs on a desktop PC running either Linux or MSWindows, and client requests are made from a pool of PCs on the same Local Area Network.

During each request for resources, the access check overhead consists of the following components:

1. Client request interceptor retrieves and attaches credential to the request.
2. Server request interceptor extracts the credential from the request.
3. The credential is sent to the Access Control Server, which responds after checking the credential.

Step 3 above involves a network round-trip, as well as possible contention at the Access Control Server. Therefore, it is the likeliest source for a bottleneck, especially in a heavily-loaded system where network congestion and/or ACS overloading may lead to delayed response, and performance hit for system requests. To eval-



(a) 2 users, varying number of services

(b) 2 services, varying number of users

Figure 5.4: Time taken to reconfigure policy on mode change

uate the scalability of this part, we construct a microbenchmark to measure the response time of the ACS with an increasing number of clients, and the results are shown in Figure 5.3. The test was performed by having a number of clients bombard the ACS with requests, and measuring the response time of a particular client. The times calculated are an average of 1000 consecutive requests. All requests are made for the same service. Figure 5.3(a) shows the mean and standard deviation of the response times, while Figure 5.3(b) shows the mean, minimum and maximum response times. The mean response time (of around 230 μsecs) does not increase noticeably with an increasing number of clients. The standard deviation is quite small ($5 - 15$ μsecs), so we expect that requests will not be arbitrarily delayed due to the access check.

The other constraint for scalability of the Access Control Service is the time taken to reconfigure policies when the space switches modes. Mode changes are triggered by context changes, including events such as users entering and leaving the space and applications starting and terminating. Figure 5.4 shows the time taken to rebuild services while switching into "shared" or "collaborative" modes. Figure 5.4(a) keeps a fixed number (2) of users in the space and varies the number of services to which access is being controlled. Figure 5.4(b) considers the effect of varying the number of users for a fixed number of services. Creating the shared mode access lists involves finding the intersection set of the permissions allowed by the individual access lists to all the users who are in the space, while finding the collaborative mode permissions involves calculating the union. Both these are affected by the number of users and services in the space; however, the times taken to recompute are fairly small, in the order of tens of milliseconds. This is the dura-

tion for which the ACS will be unavailable after a change of mode. Since we expect that sessions will typically last at least a few minutes, this overhead is acceptable for our environment.

5.6 Conclusion

The Gaia Access Control System was designed to address the security requirements of Active Space environments. The objective was to balance three requirements—security, usability and implementability—to achieve a system that can be used securely in practice. We have presented the design and implementation of this system in this chapter. To summarize:

- Access control for Active Space environments is difficult because of the interaction of the physical and virtual context of these spaces, and the variety of mobile and heterogeneous devices present in such spaces.
- The Gaia Access Control System is based on the Role-based Access Control model. Three types of roles are recognized to simplify administration in such environments. Space roles are used for access control enforcement at runtime.
- Space modes are used to indicate the current state of the space. In individual mode, access control is similar to the traditional role-based access control. Three different multi-user modes represent different types of group activity in the space, and provide support for different types of collaboration between users.
- While user authentication is not part of this research, storage and use of credentials pose some interesting problems in practical systems. Location-tracking systems allow us to mitigate this problem in certain situations.
- A qualitative performance evaluation describes the typical usage scenarios in such environments, and micro-benchmarks are used to test for scalability and sufficient response time to be usable in the Gaia environment.

6 Usability Considerations

Human factors considerations are important in designing practical security systems. Usability was an important design goal for this access control system, and in this chapter, we describe some of our attempts to achieve and evaluate this objective. The usability features of the design are described in Section 6.2. After designing the system, we conducted some user studies for administrative tools for this environment. We describe this study and the results in Section 6.3. Finally, we developed a framework to provide feedback about access control decisions to end-users, and we describe this framework in Section 6.4. But first, we provide some background on security and usability in Section 6.1.

6.1 Security and usability

Usability has been a major factor in user perception of system quality ever since the introduction of interactive systems [DHI78]. With computing blending into the background and becoming pervasive in everyday applications, a new community of non-technical users will develop around these systems. Good Human-Computer Interaction (HCI) is imperative if these systems are to become popular. Usability is of particular importance in ubiquitous computing, since end-users in these environments may have little or no access to computing support or training.

Usability and security have traditionally been viewed as being at cross-purposes, and "secure" systems are expected to be hard to use. However, we think that this is not necessarily true. While security mechanisms may restrict user actions (to an authorized subset of all possible actions), this need not make the system harder to use. By preventing unauthorized actions like network attacks or (unauthorized) memory access, effective security measures can prevent malicious attacks and system crashes, thereby improving system availability to legitimate users. Thus, securing the system improves usability, as a system that is unavailable is not very usable. Conversely, systems designed with the target users in mind are more likely to be operated correctly and avoid security lapses due to human error. Inappropriate usage of security mechanisms has been the cause of many security failures; more

consideration to human factors while designing security systems would help to improve security in practice. Thus, security and usability do not need to be traded off against each other.

While designing the Gaia access control system, we were guided by the principles of designing secure user interaction. Yee [Yee04] identifies some techniques for aligning usability with security. Zurko et al. [ZS96] describe an RBAC system that was designed with usability and security as equally important goals. The main features are an ability for administrators to query the system about the effect of policy changes before applying them. We describe the usability aspects of our access control system design below.

6.2 Design for usability

Configuring security for an Active Space environment can be complicated due to the large number of heterogeneous devices in the space and the variety of applications for a particular space. The same space can be used by different users for different applications with very different security requirements. Users bring their own devices into the space, and these devices become temporarily part of the space. However, the owners of these devices will wish to control who has access to them. The security administrator has to configure policies that represent the desired security objectives. While designing security systems for such environments, it is important to consider the practical implications of the design decisions, so as not to end up with something that is theoretically secure but impossible to configure in practice.

To achieve this objective, we paid particular attention to administrative usability while designing the Gaia Access Control system. A key feature of this process is to recognize the ways in which such spaces are typically used, so that administrative tasks can be simplified. We describe here the usability implications of various design decisions.

- Decentralized administration: We divide the administrative functions between a *system administrator* and a *space administrator* to better represent the reality of Active Space environments. We expect that Active Spaces will be part of a larger system, for example, a room in a University department or corporation, and will have to be bound by the security policies of the parent organization. The system administrator is responsible for configuring this overall policy. While the permissions allocated at the system level may be more generic ("students can access printing services in their departments"),

administrators in the specific laboratories will typically restrict this access further. An Active Space typically contains a lot of equipment, much of it being scarce and shared by different groups of users. Having a space administrator focus on permissions to these devices makes it possible to accurately represent the desired policy. This decentralization makes the system more scalable in practice. The usability principle here is to provide a suitable level of abstraction for a space administrator to configure access permissions within a space.

- Support for DAC and MAC policies: Users will want to bring their own devices, such as laptops or PDAs into an Active Space. While they can use devices in the space, they may not wish to have their laptop accessible by all others in the space. Thus, they need to be able to configure policies to their own devices. This is supported by allowing users to configure DAC policies for their personal devices. This does not require administrator intervention each time a new user-owned device enters the space. This is similar to the division of functionality between the system and space administrator—the user (device-owner) and space administrator are each responsible for "their" part of the system, and follows the principle of using "appropriate boundaries" [Yee02] between actions, i.e. boundaries that matter to users.

- Application roles: Computing in Active Spaces is more task-oriented than on traditional desktops. The access control model reflects this, by allowing policies to be specified for a particular application. Application roles can describe the permissions required to run a particular application. The space administrator can install the policies for the application as a one-time operation when it is installed in the space. Specifying policies in relation to applications follows the path of least resistance [Yee02], being the most natural way to perform this task.

- Ad hoc groups: We argue that group support in traditional operating systems is under-utilized because of the difficulty in forming and managing groups. Group creation in Unix, for example, is a relatively heavyweight process, requiring system administrator intervention. Similarly, changing group membership requires root privileges. While users can belong to multiple groups in Unix and Microsoft Windows operating systems, these features are often poorly-understood by users. We provide a more automatic creation of groups based on the presence of workers in an Active Space. We also support different types of groups, to map more closely to the different types of collaborative activities that users in such spaces undertake. Any mode that requires users to delegate their permissions (the "supervised" or "collaborative" modes) re-

quires explicit authorization, in an attempt to reduce inadvertent leakage of permissions.

While this design was guided by principles of user-centered design [ZS96], good administrative tools are still essential. The next section describes work on evaluating administrative tools.

6.3 Security administration tool

To evaluate the usability features of our design, we used some principles of user-centered design to build and evaluate administrative tools for this system. This work was performed in collaboration with Yong Liu and Kay Connelly from Indiana University. Yong Liu developed a GUI administrative toolkit for the Gaia access control system, and we performed some user studies to evaluate the administrative tools [LSC03]. The studies helped us to identify some problems with the toolkit, as well as pointing out some useful guidelines for security administration tools. We present our experiences with the user evaluation of this access control management toolkit.

Systems security failures are often the result of misconfiguration of the security mechanisms. System administrators are not always security experts, and security systems can be complicated and hard to understand. This risk is particularly important in the case of Active Spaces, as these environments are relatively new, and users and administrators are not yet very experienced. One of the problems is the lack of good tools for administrators. It is often hard to understand the effect of configuration changes—what appears to be a minor change may have ramifications on other parts of the system that are not obvious. Incomprehensible security measures are often left unconfigured, and used in the "default allow" open mode, just because they are too hard to configure correctly.

The security administration toolkit was developed to address these difficulties. As described earlier, we expect that each Active Space will have a *space administrator* whose task it is to configure the security for the space. We also expect the space to be a part of a larger system, managed by a *system administrator*. The toolkit is designed for these administrators. Security management for such environments essentially consists of managing the user-role assignment and the role-permission assignment for the *space roles* that are valid in this space. Space roles are created to perform tasks in the space. Administrators do not assign users to space roles directly, but map system roles to them, while making sure that rights amplification does not occur. Application roles represent functionality required for tasks in the

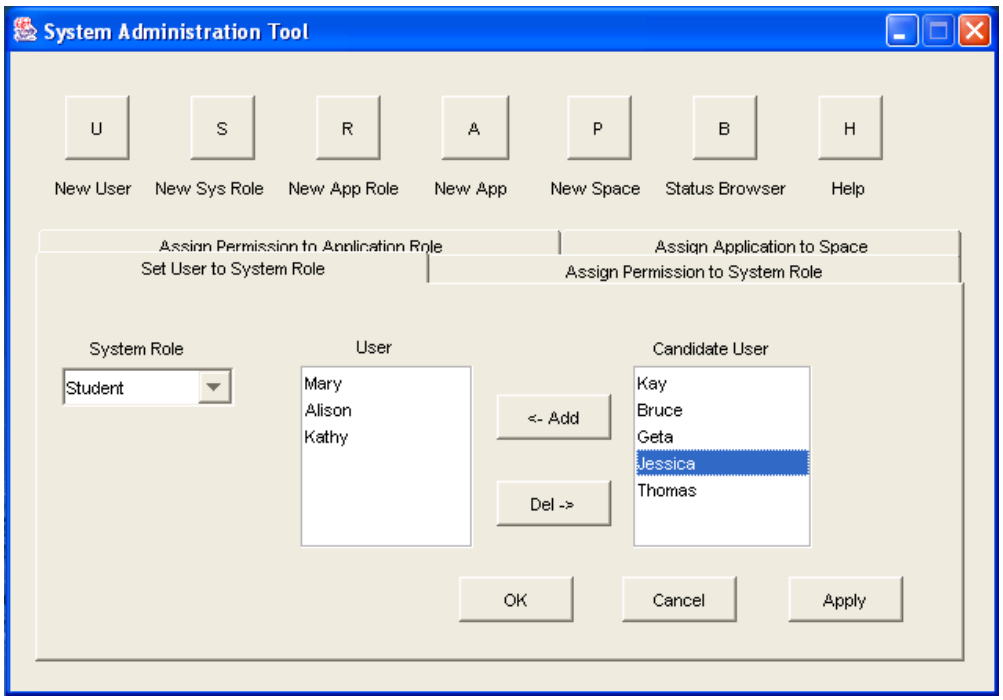


Figure 6.1: Screenshot: first version of admin tool

space. The space administrator maps application roles into space roles when the application is installed in the space.

The toolkit consists of two graphical components, one for system administrators, and one for space administrators. The system administration component allows administrators to add and remove users from the system, and assign system roles to users. It also allows administrators to add new spaces to the system. Administrators can install applications in the system and create application roles. The space administration component allows for mapping the roles. The tools save their data into files that can then be used by the system for access control enforcement. Figure 6.1 shows a snapshot of the system administration component.

6.3.1 Goals of user study

We performed a limited user study to evaluate this administrative tool. The tool was designed to help administrators navigate the process of configuring permissions for an Active Space; the goal of the study was to observe the effectiveness of the design. Another goal was to verify that the access control architecture was reasonably comprehensible to administrators who would need to configure it.

6.3.2 Design of user study

The four basic ways of evaluating user interfaces are: automatically (usability measures computed by running a user interface specification through evaluation software); empirically (usability assessed by testing the interface with real users); formally (using exact models and formulae to calculate usability measures); and informally (based on rules of thumb and the general skill, knowledge and experience of the evaluators) [NM94]. With the current state of the art, automatic and formal methods are not good enough. The most commonly used methods are probably empirical methods such as user testing. Inspection methods are a way to "save users", as real users can be difficult or expensive to recruit for testing all aspects of an evolving design. Studies of usability inspection methods [JMWU91; KCF92; Rii00] have discovered that many usability problems are overlooked by user testing, but that user testing also finds problems that are overlooked by inspection. This suggests that the best results are achieved by combining empirical tests and inspections.

We used a combination of Cognitive Walkthrough [WRLP94] and usability testing [Rub94]. Cognitive Walkthrough is an example of the inspection method. It has users, or teams of users, explore the system. It uses a detailed procedure to simulate a user's problem-solving process at each step in the human-computer dialogue, checking to see if the simulated user's goals and memory for actions can be assumed to lead to the next correct action. It is used to predict usability problems with an interface. Usability testing has users perform a set of tasks using the system. It has been used to obtain two kinds of usability measures: performance measures, which measure the functionality of a product, and preference measures, which indicate how much the users liked the product [NL94].

6.3.3 Participants

Participants in our evaluation were mostly graduate students in Computer Science, at the University of Illinois or at Indiana University. Most of them had at least some familiarity with the Gaia system. They were less familiar with the Gaia security architecture. The Cognitive Walkthrough had one team of 3 users, who were all familiar with the administrative tool and the Gaia access control system. The usability test had four participants, all graduate students in Computer Science. Half of them were working with the Gaia project. Of the others, one had no prior exposure to Gaia or the Access Control system.

6.3.4 Cognitive Walkthrough

In the Cognitive Walkthrough, a team of analysts explored a functional prototype of the administrative tool. They were given a description of 19 task scenarios that covered all the functionality of the tool. Examples of these task scenarios include adding roles, assigning permissions to roles and mapping system roles to space roles. The participants selected 5 of these scenarios as being the most representative and covering all the tool functionality, and inspected these in detail. They answered questions after the exploration, and this discussion identified some mistaken assumptions or misleading information from the interface. There was no time limit for this analysis. The details of the scenarios and the analysis checklist are attached as Appendix B. This process identified some usability problems and some desirable, but missing, features. We present all the results in Section 6.3.6.

6.3.5 Usability Testing

A total of four participants at the University of Illinois and Indiana University were given a set of tasks to complete using the prototype system. Direct observation of the participants performing these tasks, including recording using a videocamera, was used to collect extensive data. The tasks assigned were representative of typical administrative activities, and required the participants to combine several functional operations to achieve an outcome. For example, one task scenario asked the user to add a new student to the system and grant this student access to some applications. The participant had to decompose this into a sequence of actions: add the new user, assign a system role to the user, grant necessary permissions to the system role. The tasks were performed using a fully functional prototype of the administrative toolkit. Details of the tasks and materials used in the usability test are listed in Appendix C.

Participants also completed pencil-and-paper tests designed to gather information about the interface terminology. They were given a background questionnaire, two terminology questionnaires (to be completed before and after performing the tasks) and a post-test questionnaire. The terminology test was based on screenshots of the tool interface.

6.3.6 Results

Cognitive Walkthrough identified usability deficiencies in five components of the system, involving adding users to system and space roles, installing applications,

mapping roles, and browsing the system role configuration. Some of the problems were due to a lack of sufficient information about available functionality, indicating a need for more on-line "Help". Apart from the specific problems identified, the evaluators felt that more feedback to the operator about the result of actions performed would make the system more usable.

Usability Testing identified a similar number of problems. Some of these problems were identified during the test itself, in the form of participants being unable to complete tasks satisfactorily. Others were detected via the questionnaires, when it was discovered that participants were unclear about some of the tasks they had been asked to perform. Some of the problems were due a terminology mismatch between the tool developers and users. Users also found it hard to understand the concept of "mapping" roles. This was especially noticed amongst participants who were familiar with the RBAC model, and so did not pay much attention to the instructions provided. However, RBAC does *not* have the concept of mapping roles. This mainly indicates a need for better explanatory material.

Both methods noticed some of the same problems. These were of three kinds: lack of feedback after the user performed an action, some missing functionality that users expected and some ambiguous labels in the interface. The Cognitive Walkthrough also came up with some suggestions for improving the interface, by providing "undo" functionality and re-organizing some of the GUI layout.

The problems found by usability testing alone were specific problems users encountered while trying to perform the specific tasks. Users did not request any more functionality, probably because they did not have enough experience with the tool and environment to identify improvements. Since Cognitive Walkthrough is a more open-ended method, it is useful for pointing out such design problems and suggesting improvements. The two methods, thus, complement each other.

6.3.7 Lessons learnt

The study we completed was an initial attempt to identify requirements for security administration tools for Active Spaces. Based on the results from the tests, the tool was improved to incorporate usability enhancements. The main improvements were revising the terminology, using a more graphical representation and improving the visual layout and re-organizing the functionality.

The revised tool used hyperbolic trees [LRP95] to display the role hierarchy. Hyperbolic trees, which are a dynamic representation of hierarchical structure, are an effective way to display complex trees clearly. They use a focus+context technique

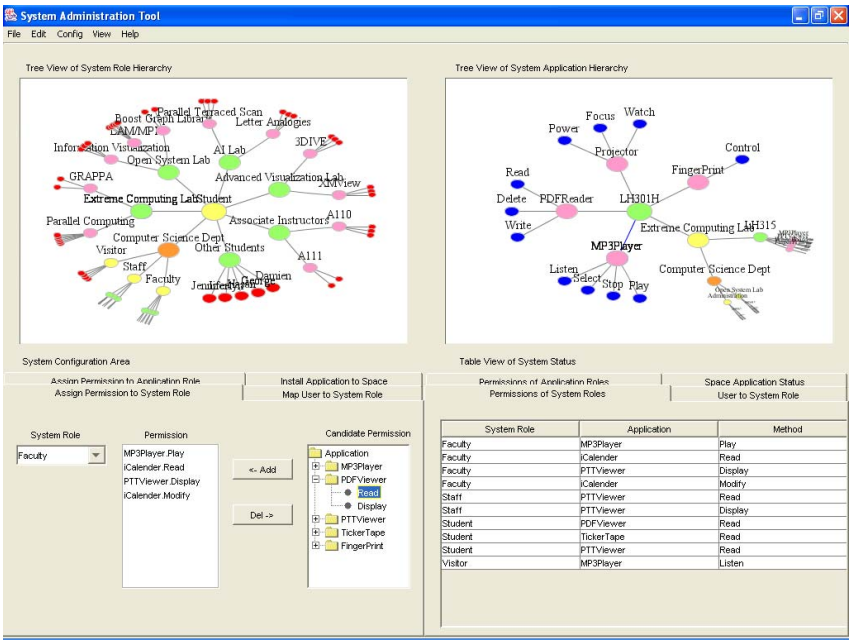


Figure 6.2: Screenshot of improved administrative tool

to provide a good integration of detail and context on small displays. A snapshot of the improved tool is shown in Figure 6.2. We have not yet conducted user tests of this version of the tool.

Further work in information visualization techniques is also likely to be useful. One of the problems for administrators is the inability to predict the effect of a configuration change. Good information visualization can highlight these changes. Providing this visualization along with the ability to view the effects of changes without actually applying them would be helpful to system administrators.

While the user evaluation was conducted to evaluate a specific tool, some of the principles identified apply to security administration tools in general. System administrators typically perform a few types of tasks many times over, and appreciate ways to automate or script such tasks. While a GUI administrative tool is useful for visualizing system state or making small changes, administrators often want a more programmable way to make many changes at once. Another difference is that administrators are more likely to look for and read the documentation than a casual user. Similar results were found by in the design of Adage et al. [ZS96], an authorization service for distributed applications, where security and administrator usability were considered equally important design goals.

As lack of feedback appears to be a common problem with system usability, we proceeded to investigate the issue further. The *Know* system, described next, was

an attempt to provide feedback about access control to end-users.

6.4 *Know*—policy feedback for users

Users in ubiquitous computing environments typically interact with a plethora of computing, communication or I/O devices in their vicinity in many ways—voice, gestures, and traditional keyboard-and-mouse input being some of them. Different sets of users are allowed access to different subsets of resources, and these permissions may change depending on contextual information such as the time of day, the current activity, or the set of people involved. In such an environment, it may not be clear to a user why he or she was denied access to certain resources. Thus, informative feedback about why access was denied becomes very important if the system is to avoid annoying users with apparently-arbitrary restrictions. However, unrestricted feedback about who is allowed to do what in the system could itself compromise system security and privacy; therefore, policies need to be protected against inadvertent disclosure. Apu Kapadia and I explored this space of policy feedback using the *Know* [KSC04] system. *Know* attempts to provide useful feedback when access is denied, while maintaining policy confidentiality with the help of meta-policies. Cost functions are used to rate the value of the different possible feedback options.

Good security feedback is particularly important for ubiquitous computing systems, such as Active Spaces, for a variety of reasons:

- Ubiquitous computing is an area of active research [RHC+02; JFW02]; new systems and modes of applications are being developed. While these systems are still being used in experimental ways by researchers, application developers and early adopters, good security feedback will help direct secure application design. Security is hard to retrofit into existing applications.
- Ubiquitous computing environments, currently most prevalent in academic and research environments, are expected to percolate into everyday use, with a majority of non-technical users. In both these situations, feedback is important for usability, since users either disable or work around security mechanisms that seem incomprehensibly obstructive.
- Ubiquitous computing systems can be more confusing to users than traditional distributed systems due to the inherent dynamism (mobile users and devices may enter and leave the system), the large number of devices and the context-sensitive environment—without adequate feedback, it can be dif-

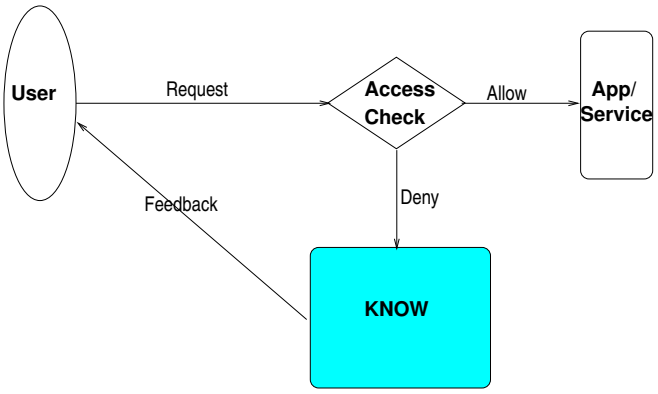


Figure 6.3: *Know* architecture

ficult to tell whether access was denied due to a bug in the system (especially in experimental systems) or due to user permissions changing in response to non-obvious changes in the context.

These reasons led us to explore the ideas that led to the development of *Know*.

6.4.1 *Know* architecture

The features of the access control model that are relevant for *Know* are the following. Users get assigned to a "space role" on entering the space and authenticating themselves, and access control enforcement is performed in terms of these roles. The Access Control system maintains the policy for each resource in the space. Policies are represented as boolean formulae that control access to a particular resource. Permissions may depend on both user identity (or role) and contextual information, such as the number of users in the space or the currently scheduled activity. Thus, permissions available to a user at any point in time depend on a variety of factors, and good feedback about access control decisions is very important.

Know is implemented in the form of a feedback component that can augment the access control system, as shown in Figure 6.3. The access control system intercepts all requests and checks them against the system policy. If authorized, they are allowed to proceed. If not, they are forwarded to *Know*, which prepares a feedback message for the user. Feedback consists of a list of alternative conditions under which this access is permitted. For example, a user trying to access a noisy printer in a room during a meeting may receive feedback such as "If there is no meeting, you will have access." Alternatives may not always be available, either due to policy or computational resource constraints. In this case, the standard "Access is

Policy:

$$
\begin{aligned}
R &: P \\
P &\leftrightarrow P_1 \vee P_2 \\
P_1 &\leftrightarrow \mathsf{User.role} = \mathsf{Professor} \\
&\quad \wedge \mathsf{User.department} = CS \\
P_2 &\leftrightarrow \mathsf{User.role} = CIA
\end{aligned}
$$

Meta-Policy:

$$
\begin{aligned}
P_1 &: \mathsf{User.department} = CS \\
P_2 &: false
\end{aligned}
$$

Figure 6.4: Example policy

denied" message is provided.

We describe this system in more detail below.

Policy representation

UniPro [YWS03] provides a generalized framework to model the protection of resources, including policies, in trust negotiation. It allows policies to be treated as resources in the system, and allows the specification of policies to protect them. *Know* uses the UniPro notation for writing meta-policies that protect the policies to decide what feedback can be provided to a user. We demonstrate the notation with an example:

Example: The access policy of an electronic door lock might allow access only to Computer Science professors or members of the CIA. When a person is denied access to the room, feedback of the form, "If you are a CS professor or a member of the CIA, then you will have access to this room" is potentially dangerous. Collaboration between the Computer Science department and the CIA could be sensitive information. Outsiders may also glean intelligence information about where CIA members meet. Clearly, we may not want to reveal parts of this access rule. Feedback of the form, "If you are a professor in Computer Science, then you will have access to this room" may be acceptable. A meta-policy would control this flow of information to denied users. Formally, we can represent the policy and meta-policy as shown in Figure 6.4.

Access control policies here are represented as boolean formulae. A policy definition includes two types of expressions. An expression of the form $O : P$ means

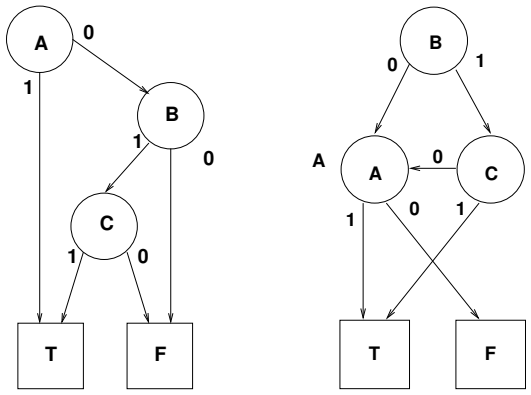
that an object O is protected by policy P, where policies themselves can be objects (since policies may be protected by meta-policies). An expression of the form $P \leftrightarrow E$ means that the policy P is defined by expression E. Expressions can contain both atomic propositions (e.g., User.department $= CS$) and references to subpolicies (e.g., $P \leftrightarrow P_1 \vee P_2$, where P_1 and P_2 are defined subsequently). The access policy for the room is $R : P$, which means that access to the room R is protected by policy P. P is defined as the disjunction of policies P_1 and P_2. P_1 is the policy, "User must be a professor in Computer Science." P_2 is the policy, "User must be a member of the CIA." Hence the policy P to access the room is "User must be a professor in Computer Science or the user must be a member of the CIA."

We make two assumptions here. First, we assume that any logical dependencies between atomic propositions are captured within the policy. For example, a policy may contain atomic propositions User.isAdult and User.isMinor. We know that User.isAdult $\Leftrightarrow \neg$User.isMinor, and hence feedback of the form "If you are an adult and a minor, you will have access" would be absurd. Such inconsistencies are avoided by either replacing occurrences of User.isMinor by $\neg$User.isAdult or by adding the logical rule User.isAdult $\Leftrightarrow \neg$User.isMinor to the policy. This will avoid any inconsistencies in feedback. The second assumption we make is that all references to an atomic proposition a are protected by the same meta-policy.

It is important to note that in all our examples we are careful to provide feedback as "If... then" clauses. This is important for policy protection. Feedback of the form, "Only professors may access this room" gives more information than "If you are a professor, then you will have access to this room." If both types of feedback were allowed, the user may infer from the latter feedback that there is a protected policy not being revealed. Hence, if feedback is consistent in its use of "If... then" clauses, users will not gain any extra information about protected policies.

Ordered Binary Decision Diagrams

Ordered binary decision diagrams (OBDDs) [Bry86] are a canonical-form representation for boolean formulae where two restrictions are placed on binary decision diagrams: the variables should appear in the same order on every path from the root to a terminal, and there should be no isomorphic subtrees or redundant vertices in the diagram. A binary decision diagram is a rooted directed acyclic graph with two types of vertices: terminal and nonterminal. Each nonterminal vertex v is labeled by a variable $var(v)$ and has two successors, $low(v)$ and $high(v)$. We call the edge connecting v to $low(v)$ the 0-edge of v (since it is the edge taken if $v = 0$) and the edge connecting v to $high(v)$ the 1-edge of v. A single formula may



(a) Variable ordering: a, b, c

(b) Variable ordering b, c, a

Figure 6.5: Example OBDDs for $a \vee (b \wedge c)$

be represented by multiple different OBDDs based on the order that variables in the formula are tested; however, given a particular variable-ordering, the OBDD structure is fixed. This is the canonical form for that variable-ordering. Figure 6.5 gives an example of two OBDDs that each represent the simple boolean formula $a \vee (b \wedge c)$. The first is the canonical-form OBDD for the variable-ordering a, b, c and the second is the canonical-form OBDD for the variable-ordering b, c, a. To test for satisfiability, we start at the root node and test whether the variable at the root is *true* or *false*. If it is *false*, we follow the 0-edge, and if *true*, the 1-edge, and repeat this process. Eventually we reach either the T-node or the F-node (also called the 1-node and 0-node, respectively). If we reach the T-node, then the given assignment satisfies the formula; if we reach the F-node, it does not. For example, applying the assignment $\langle a = false, b = true, c = false \rangle$ to either of the OBDDs in Figure 6.5 tells us that the formula is not satisfied. We use OBDDs because they are a compact and graphical representation of boolean formulas. This allows us to use cost functions and shortest path algorithms to find conditions of satisfiability that are of "least cost" to the user.

Know stores access control rules as OBDDs, and can efficiently search these OBDDs for paths that satisfy the rules. When access is denied, the OBDD can provide information about alternate paths that would allow access. *Know* provides information to the user about such paths as feedback. The number of nodes in an OBDD can be exponential in the size of the boolean expression, but there are several heuristics to find an ordering that reduces the size of the OBDD, and in practice, boolean functions usually have a compact OBDD representation. As mentioned earlier, *Know* provides feedback only if it can be done with acceptable overhead.

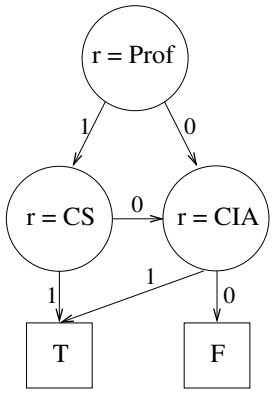


Figure 6.6: OBDD for the example policy

Figure 6.6 shows the associated OBDD for the example policy shown in Figure 6.4. The meta-policy P_1 : User.department $= CS$ indicates that the policy P_1 may be revealed only to subjects in the Computer Science department, while the meta-policy $P_2 : false$ does not reveal P_2 under any circumstances[1]. For example, a denied student in Computer Science would receive the feedback "If you are a professor in Computer Science, then you will have access to this room," while a student in Civil Engineering will be informed, "Access is denied." In either case, no policy information involving the CIA is revealed. We will describe the application of meta-policies to OBDDs using cost functions as described below.

Cost functions

There may be many paths from the root to the *True* node, each representing a different set of conditions under which access to the resource is allowed. However, they are clearly not equally *useful* to the user. For example, telling a user who is a Student that access is allowed to a Professor may not be particularly useful, while telling her that a Student is allowed access if a labsitter is present is more useful. Cost functions are used to represent the relative usefulness of the available feedback options.

Selecting a path from the list of feedback options implies that all propositions along that path are satisfied, i.e. the path only uses outgoing *True* edges from a node representing a currently true proposition or outgoing *False* edges from nodes representing a currently false proposition. Obviously, no path has this property, or access would have been allowed, and *Know* would not have been invoked. The cost

[1]In our examples we omit meta-policies of the form $P : true$ for clarity. In practice however, all meta-policies may be assumed to be of the form $P : false$ unless a meta-policy is explicitly specified.

of a particular feedback option thus is the cost of *changing* propositions to make this path represent the system state—*i.e.* changing a currently true proposition to false if the path follows the outgoing *False* edge and vice versa. We use a cost function to represent the difficulty of changing the value of a particular proposition.

A simple cost function could assign a uniform cost to changing any proposition. The cost of a particular feedback option then is simply the *number* of propositions that need to be changed. However, all propositions are not equally difficult to change, so this may not be a very realistic cost function.

A better cost function, in some situations, would be to recognize that users cannot change their role. This effectively assigns infinite cost to any proposition representing role information. Other propositions could be considered equally difficult, as in the previous case. Now, feedback options that require role changes will simply be eliminated (as having infinite cost), and the remaining options (that require only context changes) will be ranked by the number of required changes.

Our current system assumes a system-wide cost function. We expect that different systems will have different cost functions. For example, a student lab may find feedback asking users to come back at night useful, whereas other office environments may not. Per-user cost functions are an option to explore, since different user preferences may result in different opinions on the usability of feedback options.

We provide some notation and a formal definition of feedback.

We use the notation $S \models P$ to indicate that a policy P is satisfied under the atomic propositions specified by S. For example, we could have $S = \{\mathsf{Context.meeting} = false, \mathsf{Context.workingHours} = true\}$. In the notation $S[a] \models P$, $S[a]$ is the set of atomic propositions in S along with any update provided by a. In our example, $S[\mathsf{Context.meeting} = true]$ = {Context.meeting $= true$, Context.workingHours $= true$}. This notation naturally extends to a set of updates, e.g., $S[A]$, where A is a set of atomic propositions. Let C be the set of atomic propositions relating to the context of the system and U be the set of atomic propositions specific to the user (identity, role, etc.). Given a policy P and a user U, the user is granted access when $C \cup U \models P$, and denied access when $C \cup U \not\models P$. In essence, if $C \cup U \not\models P$, then a set of updates X such that $(C \cup U)[X] \models P$ constitutes a feedback option to the user.

To formalize the notion of feedback, let $\Pi = \{\pi_1, \ldots, \pi_n\}$ be the set of paths from the root node to the *true* node in the OBDD of P. Let π_i' be the set of atomic propositions that appear in $\pi_i \in \Pi$ and whose truth values differ in $C \cup U$, i.e., the set of propositions that must be changed (or a set of updates to the state) for the policy to

be satisfied. Let $\mathcal{F} = \{\pi'_1, \ldots, \pi'_n\}$. Note that $(C \cup U)[\pi'_i] \models P$ for all $\pi'_i \in \mathcal{F}$. We define any subset F of $\mathcal{F}$ to be the $feedback$ offered to the user. In other words, each *feedback option* f_i in the *feedback* F corresponds to a set of atomic propositions the user must change to be granted access. $\mathcal{F}$ is the set of all possible feedback options available to the user. Since $\mathcal{F}$ can be very large, our primary goal is to find a way to offer the user only a few relevant feedback options in $\mathcal{F}$. We do this through the use of cost functions. The cost function assigns a cost to each $f \in \mathcal{F}$, and returns the k lowest-cost feedback options, where k is a tunable parameter.

A naïve cost function could assign the same cost to each change, in which case the user would be given feedback with the least number of changes that need to be made to access a resource. For example, we could sort the elements f_i of $\mathcal{F}$ in ascending order of $|f_i|$ (number of atomic propositions in f_i) and return the first k choices. However, changing roles might be more difficult than changing context. For example, a Student may be able to come back at a later time, but it would be extremely difficult to acquire a Professor role. This suggests the use of more sophisticated cost functions.

We need to define an appropriate cost function that is applied to edges in the OBDD as edge weights. Using these weights we can use shortest path algorithms from the root to *true* to provide feedback with lowest total cost. Running Dijkstra's algorithm gives us a path with lowest total cost in polynomial time. There are several proposed algorithms for k shortest paths for graphs. Eppstein [Epp94] presents an algorithm that computes k shortest paths in time $O(m + n\log n + k)$, where n is the number of vertices, and m is the number of edges in the graph. This is the best known bound for k shortest paths in directed acyclic graphs. Since an OBDD with n nodes has $2n-4$ edges (two children for each node, except the *true* and *false* nodes), the complexity for computing the k shortest paths in an OBDD is $O(n\log n + k)$.

Let A be the set of atomic propositions in the policy P. We define a cost function $c : A \times \{0, 1\} \rightarrow \mathbb{R}^+ \cup \{\infty\}$, where $\mathbb{R}^+$ is the set of non-negative real numbers. This function tells us the cost to change an atomic proposition, an in effect, the cost to follow a 0-edge or a 1-edge for a node in the OBDD. An infinite cost disallows any changes to the current value of the proposition. When a request for access is denied, let $T \subseteq A$ be the set of propositions that evaluate to *true*, and $F \subseteq A$ be the set of those that evaluate to *false*. We define $c(t, 1) = 0$ for $t \in T$ and $c(f, 0) = 0$ for $f \in F$ since there is no cost to maintain atomic propositions that are satisfied under the current conditions ($C \cup U$), and we would like to assign non-zero cost when a user must *change* some atomic proposition. Cost functions will differ according to their assignments to $c(t, 0)$ for all $t \in T$ and $c(f, 1)$ for all $f \in F$. Now, for all $a \in A$, assign the weight $c(a, 0)$ to the 0-edge of a, and $c(a, 1)$ to the 1-edge of a. What

results is a directed acyclic graph with weights assigned to each edge. We can now apply k-shortest path algorithms to this graph to get the k lowest-cost paths, which correspond to the k lowest-cost feedback options. For small k, the running time for such algorithms is dominated by the structure of the OBDD and not k. Specifically, since we expect to have $k < n$ (for example $k = 3$ might be sufficient), the running time is $O(n\log n)$. Our naïve cost function that considers all changes to be equally expensive would set $c(t, 0) = 1$ for all $t \in T$ and $c(f, 1) = 1$ for all $f \in F$. Hence the total cost of any path is equal to the number of propositions that need to be changed under the given conditions.

6.4.2 Meta-policies

Meta-policies contain information about who is allowed to *view* parts of the policy. To honor the meta-policies, *Know* must not provide any feedback option that contains information about propositions that are forbidden by the meta-policy. This can be achieved by assigning an infinite cost to changing these propositions. As described for the "better" cost function above, this effectively disables any changes in the "protected" propositions by making these changes incur infinite cost. We describe how the algorithm above, for cost functions, can handle meta-policies.

Each meta-policy determines whether a user can read certain nodes in the policy's OBDD. Let $D \subset A$ be the set of nodes forbidden by the meta-policy. For each $d \in D$, we assign infinite cost to the edge that effects a change in the current value of d. This does two things: first, it prevents shortest path algorithms from exploring a change in d and hence does not return any feedback options that require a change in d. Second, since this proposition d cannot be changed, it will not appear within a feedback option, which includes only those propositions that must be changed. Since no atomic proposition that is precluded by the meta-policy appears in any feedback option, *the feedback given to the user honors the meta-policy*. We assume that all nodes corresponding to a particular atomic proposition a are protected by the same meta-policy, allowing us to perform such a transformation. Finding efficient ways of computing consistent feedback where references to the same atomic proposition are protected by different meta-policies is left to future work.

6.4.3 Implementation

We built a prototype of the *Know* system, and, in this section, we describe the implementation and results from a preliminary evaluation. We present results of *Know*

running with an example access control policy for videoconferencing equipment located in a kiosk within a multi-purpose business center.

The system access policy is represented as an OBDD, which is then transformed into a weighted graph that is specific to access requests. An appropriate cost function, along with the system meta-policy, is used to assign weights to the edges. Finding the k shortest paths to the 1-node of the OBDD gives us k sets of assignments to the variables that will satisfy the access control rules, and thus, describe k situations under which the particular operation is allowed.

The first step is to generate an OBDD from the system access control policy. We use the BuDDy [LN99] library, which uses heuristics for optimizing the generated OBDDs. The end result of this is an OBDD that represents all allowable ways to perform a particular action (or access a particular resource). If the requested action is permitted, *Know* is not needed. If not, *Know* attempts to find alternative paths in the OBDD that would permit the operation, *i.e.*, paths in the OBDD from the root to the 1-node.

Alternative paths are found by using the Eppstein [Epp94; Gra] algorithm to find the k shortest paths from the root to the 1-node in this OBDD. Weights are assigned to the edges of the OBDD graph based on the cost function and the current values of the user roles and context variables. Selecting a suitable cost function is site-specific—the weights assigned to the different changes will depend on the nature of tasks that are normally performed by users of the system. We provide results from the two cost functions described earlier—the naïve cost function (which counts the number of changes required) and the "useful" cost function (which treats role changes as more difficult to achieve than context changes).

Know then outputs the necessary changes that must occur to satisfy the alternative paths. It is up to the user to choose between these suggestions, and to retry the request after following the suggestion.

We illustrate this entire process with its application to a sample policy that governs the access to videoconferencing devices in the business center of a hotel. In addition to computers, the business center also contains devices such as printers, fax machines, cameras for videoconferencing and so on. The business center is located in the conference hall, and hotel guests and other members who have signed up are normally allowed to use the devices as per the security policy. The conference hall is also rented out for activities such as meetings, conferences, or receptions, during which time use is restricted to participants of this activity, as per the policy configuration by the organizers. Users present their credentials to enter the business center, in the form of a smartcard (a conference badge or a hotel room key) and

the system uses this information to restrict access and provide useful feedback. We present here the rules that affect access control to the camera for the videoconferencing system.

The basic policy is as follows:

- When no activity is scheduled for the room, supervisors, hotel guests or other registered users can use the videoconferencing equipment during the business day. Visitors are also allowed to use the facilities if an operator is present. Hotel guests may also use the system during non-business hours, but others may not.
- When an activity (such as a videoconference) is scheduled, only registered activity participants and supervisors are allowed to use the system.
- Use of the videocamera is disallowed for regular participants if the videoconferencing activity being undertaken in the conference center is labeled as confidential. However, the meeting supervisor may still turn on the videocamera if all participants have the required security clearance.
- Maintenance activities are performed by designated personnel.
- Finally, a high ambient temperature indicates some problem with the air-conditioning system, and camera use is prohibited until temperature reaches the allowed range. Similarly, overcrowding the room will violate the fire safety codes and cause access to the camera to be denied.

The meta-policy that governs feedback contains the following rules:

- Information about confidential activities is only provided to the meeting supervisor. Thus an unauthorized user trying to access the videocamera during a confidential activity will not be informed that a confidential activity is going on, but just that access is denied at that time. Similarly, feedback about the presence of uncleared users is only given to the meeting supervisor.
- Information about maintenance activities is not provided to other users.

The access control rules for this policy above are presented in Figure 6.7. In our implementation, access to the camera C is protected by policy P. Policies $P_1, \ldots, P_{10}$ describe the various rules presented above, where P_7 and P_8 are rules pertaining to the VideoConference activity. In the interest of brevity, we only present the rules relevant to the VideoConference activity in Figure 6.7.

This policy states that during a confidential VideoConference, only a Supervisor can access the camera as long as there are no uncleared users present. During

Policy:

$$
\begin{array}{rcl}
C & : & P \\
P & \leftrightarrow & P_1 \vee \ldots \vee P_{10} \\
\ldots & & \\
VC & \leftrightarrow & \mathsf{activity} = \mathsf{VideoConference} \wedge \neg(A_1 \vee \ldots \vee A_n) \\
CA & \leftrightarrow & \mathsf{Context.isConfidential} = true \\
RS & \leftrightarrow & \mathsf{User.role} = \mathsf{Supervisor} \\
NU & \leftrightarrow & \mathsf{Context.UnclearedUsersPresent} = false \\
NH & \leftrightarrow & \mathsf{Context.cameraOverheated} = false \\
NF & \leftrightarrow & \mathsf{Context.roomFull} = false \\
RP & \leftrightarrow & \mathsf{User.role} = \mathsf{Participant} \\
P_7 & \leftrightarrow & VC \wedge CA \wedge RS \wedge NU \wedge NH \wedge NF \\
P_8 & \leftrightarrow & VC \wedge \neg CA \wedge (RP \vee RS) \wedge NH \wedge NF
\end{array}
$$

Meta-Policy:

$$
\begin{array}{rcl}
\ldots & & \\
CA & : & \mathsf{User.role} = \mathsf{Supervisor}
\end{array}
$$

Figure 6.7: Example policy used for evaluation

a non-confidential VideoConference, any Participant or Supervisor can access the camera. The room must never be Overheated or Full during a VideoConference. The second rule in the meta-policy states that only Supervisors will be made aware of Confidential activities (or the lack thereof). Hence if an ordinary user is denied access to a camera, the user will not be told that there is a confidential conference in progress (this information itself is deemed sensitive). Since there can be only one activity at any given time, the policy specifies $VC \leftrightarrow \mathsf{VideoConference} \wedge \neg(A_1 \vee \ldots \vee A_n)$, where $A_1, \ldots, A_n$ are the remaining activities.

The OBDD generated by the above policy has 17 variables and 35 nodes (in contrast, a binary decision tree would have at least 2^{17} nodes).

To evaluate *Know*, we try to access the videocamera under a variety of situations, and present the suggestions provided by *Know* using each of the two cost functions described earlier, which we designate as the "naïve" and the "useful" cost function. Since the useful cost function just restricts information about role and activity change, feedback from the useful cost function will just be a (more useful) subset of the feedback from the naïve cost function. We describe some of the experiments below for $k = 4$. The run-time overhead for *Know* to find these suggestions was negligible—in the order of milliseconds. Since OBDDs are just a representation of the access control policy, they can be constructed ahead of time and only need to

be re-computed if the policy changes. Assigning weights to the edges of the OBDD is performed each time a request arrives, since the weights depend on the current values of the context variables and user credentials. Since *Know* runs only when access is denied, it has no performance overhead on successful requests. We now describe the situations and results in detail:

- A Visitor tries to use the camera during business hours, but no Operator is present. There is no activity in session. With the naïve cost function, *Know* suggests that the user come back a) as a HotelGuest b) as a RegisteredRoomUser c) when an Operator is present, or d) as a Supervisor. The useful cost function suppresses the suggestions involving a role change, and only advises the user to come back when an Operator is present. This simple example illustrates the basic functionality of *Know*.

- If a HotelGuest tries to use the equipment during working hours when the room is too hot and there is no activity in session, *Know* correctly suggests that the user try again a) when room is not Overheated b) when room is not Overheated and it is out of business hours and c) when room is not Overheated and as a RegisteredRoomUser instead of a HotelGuest, or d) when room is not Overheated, as a Supervisor, instead of a HotelGuest. The useful cost function only offers the first two suggestions because it does not recommend role changes. Clearly, the only change *required* is for the temperature to be reduced, but *Know* does not presently restrict suggestions that are subsets of others. This may be useful in some situations.

  Maintenance operations are allowed even in overheated conditions, and a straightforward search through the policy might have offered the suggestion to try coming back as a MaintenanceWorker. However, the system meta-policy forbids the disclosure of information about maintenance permissions, so this option is correctly ignored by *Know*.

- During a Confidential videoconferencing activity and regular working hours, if a Participant tries to access the videocamera when users without the required security clearance are present, the naïve cost function suggests the user come back a) as a HotelGuest when no activity is in progress, b) as a RoomUser when no activity is in progress, c) as a Visitor when no activity is in progress, or d) as a Supervisor when no activity is in progress. The useful cost function does not offer any feedback, because there is no useful option for the Participant.

  One possible suggestion is to inform the user that this operation is not permitted during a confidential activity and to suggest re-trying when no confiden-

tial activity is being undertaken, but the system meta-policy precludes any information about confidential activities from being revealed, so this suggestion is not offered. Note that it is possible for users to correlate feedback from different sessions to infer the *existence* of hidden atomic propositions. For example, the presence or absence of a confidential activity results in differing feedback. Care must be taken while writing the policies and meta-policies to prevent the leakage of the *identity* (e.g., confidential activity) of the atomic proposition.

- If a Supervisor tries to use the camera when the room is reserved for a confidential VideoConference and uncleared users are present, the uniform cost function suggests that the user come back a) after changing the activity type to be non-confidential b) when no uncleared users are present, c) when there is no activity scheduled, or d) as a Participant after changing the activity type to be non-confidential. The useful cost function suggests the first three options. Note how the Supervisor is given feedback regarding Confidential activities, as opposed to a Participant in the previous scenario.

While the above examples are fairly simple, they validate our hypothesis that *Know* can provide useful information about alternatives when access is denied, that it can do so without compromising privacy or confidentiality requirements of the security policies, and that this can be achieved with negligible performance overheads. *Know* provides a useful framework for further studies on cost functions and usability.

6.5 Conclusion

Human factors are important for the correct use of security mechanisms. They are particularly important in ubiqutious computing environments, because these environments are expected to be non-intrusive. The Gaia access control system was designed with usability in mind. We identify two aspects of the Gaia access control System where usability concerns are important. An important, and understudied, aspect is usability for *administrators* of such environments. We performed user studies on an administrative tool designed for the Gaia access control system to evaluate its effectiveness. The user studies indicated that visualization tools to display the state of the configuration would be useful, as well as other methods for feedback to administrators about the results of their actions.

For *users* of the space, the dynamic environment and permissions that change with system context can be confusing to end-users. To help with this, we developed a

framework to provide users with feedback about access control decisions. *Know* is an initial attempt to balance the requirements of feedback and policy protection. Our results indicate that this is a feasible approach. Useful feedback can be provided at reasonable cost, while maintaining policy confidentiality.

While the feedback system does not reveal any information that is protected by the meta-policy, we are concerned that having to configure meta-policies as well as policies may make the system administrator's job more difficult. We plan to study this further and identify meta-policy guidelines for administrators, based on the properties they wish to provide.

7 Conclusion

We have presented and evaluated an access control system for Active Spaces. We now draw some overall conclusions and outline future work. In Section 7.1, we present a summary of the thesis research. Section 7.2 enumerates the contributions of the thesis work. Finally, Section 7.3 places the research in context, points out some of the remaining problems and avenues for future research.

7.1 Summary

Ubiquitous computing environments promise exciting new applications, but pose new security challenges, which must be addressed before these environments can be widely deployed. Access control is a basic security mechanism that is required if a system wants to provide any security guarantees. In this dissertation, we have described an access control architecture for a class of ubiquitous computing environments known as Active Spaces.

Active Spaces are physical spaces that contain a variety of heterogeneous computing and communication devices, with software infrastructure to integrate these resources into a unified and programmable environment that users can interact with. The main challenges posed by the Active Space environment for access control are due to the context-sensitive and dynamic, heterogeneous and device-rich nature of the environment. These environments are also typically used in different ways from traditional distributed computing environments—groups of users typically collaborate in using the space for particular applications, rather than a personal computing mode of usage. Another important difference is in the nature of users—as computing becomes more pervasive, it is no longer feasible to expect users to undergo specialized training to use these systems, so the "naïve user" is much more common. Making these systems easy-to-use is important if they are to become truly ubiquitous. Usability is especially important for security systems, as misconfigured security mechanisms have been responsible for a variety of security compromises in computer systems.

Access control is a basic security mechanism, and required to provide any other

security guarantees. As new applications for Active Space environments are still being developed, the modes of usage may change further, requiring that access control be flexible enough to support their requirements. The main requirements for access control in such ubiquitous computing environments, therefore, are:

- Support for collaborative usage patterns prevalent in such environments.
- Ability to incorporate context into the access control decision.
- Easy-to-use, for both end-users and security administrators.
- Flexibility, to support a variety of access control requirements for different applications.
- Implementability on a heterogeneous platform, with widely-varying resource availabilities.

We have presented a model that addresses the above requirements, and developed a prototype implementation within the Gaia framework. Our model extends the Role-Based Access Control model with three types of roles to simplify policy configuration and administration. *System roles* are akin to the roles used in traditional RBAC, representing a user's permissions in the organization. Within an Active Space, a user's system role gets mapped into a *space role*, representing a subset of permissions that are valid within that particular space. *Application roles* are used to represent application-specific sets of permissions, and simplify policy management in Active Spaces, since Active Spaces are more commonly used for specific applications than for general-purpose computing. Both system roles and application roles are mapped into space roles within a particular Active Space, and access control is enforced in terms of the space roles.

We also introduce the notion of space *modes* to represent the physical context of the space, and support different types of collaboration. The three different modes for group usage of the space—shared, collaborative and supervised—represent different types of activities. *Shared* mode is for users sharing the space without necessarily working together or trusting each other. *Collaborative* mode allows a group of collaborating users to share permissions for the duration of a particular activity. *Supervised* mode is a form of limited delegation, and allows users to perform actions in the presence of a supervisor that they would not be permitted to on their own.

We evaluate this system on the criteria of expressivity, performance and usability.

7.1.1 Expressivity

Since new types of applications are still being developed for ubiquitous computing environments, access control models need to be flexible, so as to support these emerging modes of usage. At the very least, *mandatory* and *discretionary* access controls are required to control access to the shared space resources and user-owned devices that temporarily join an Active Space, respectively. Our model supports both these modes. Active Spaces operate in more decentralized ways than traditional systems. Policy specification may need to combine the centralized organizational security policy with the local space policy, or the space MAC policy may need to be combined with the DAC policy for devices brought in by individual users.

The model also allows policies to be specified per application; this is useful because Active Spaces tend to be used for particular activities, and user permissions depend on the current system context and activity. Expressing the policy in this way is a more natural fit for such environments.

To summarize, our model supports discretionary and mandatory access control policies, can incorporate contextual information, and is aware of modes of collaboration. This has been sufficient to express the access control requirements for our Active Space; we argue that it is flexible enough for future applications as well.

7.1.2 Performance

Since access control only works if all requests for resources are checked before being allowed to proceed, it can easily become a performance bottleneck. Access control must be reliable enough and introduce a low enough overhead that it not disrupt system performance. We designed the Gaia Access Control System with a goal of having a low run-time overhead. It has been continuously operational in our prototype smart room for over a year, and has been found to be acceptable to the user community. We demonstrate scalability using micro-benchmarks that measure the response time of the access control system as the number of services, roles and context variables change. We observe that the performance overhead of the access control system is negligible compared to other operations in the Gaia system.

7.1.3 Usability

Usability is an important factor in the design of practical security systems, since user errors often lead to poor security in practice. While usability for end-users is

increasingly receiving attention, we think that administrative usability is an important problem that has not received sufficient attention. Especially in Active Spaces, with their variety and large number of devices and modes of usage, configuring security policies may prove to be difficult. To address this, we performed a study to evaluate the usability of the access control model from a security administrator's point of view. The results indicate that the system is comprehensible with some training, but better administrative tools would help. Specifically, the studies highlighted the need for information visualization, and context-sensitive help in administrative tools.

The context-dependent varying of permissions can make access control decisions confusing to users of the system. To improve usability for them, we developed *Know*, a framework to provide feedback about access control decisions when permission is denied. Since the access control system has knowledge about the system policy and the current context, as well as access to the user credentials, providing suggestions about alternative ways to access system resources is relatively straightforward and can be useful. A challenge is to provide this feedback while maintaining the confidentiality requirements of the policy, which we handle by means of meta-policies that can restrict access to the policies themselves.

7.2 Contributions

The main contribution of this thesis is the design of an access control system for a class of ubiquitous computing environments. The specific contributions are:

- An access control model that incorporates system context, and can support a variety of policies, including mandatory and discretionary access control.
- Identification of, and support for, the access control requirements for a variety of collaborative modes of usage for such ubiquitous computing environments.
- An evaluation of a prototype implementation, on the basis of performance and usability.
- A technique for providing feedback about access control to end-users, to improve the usability of the system without compromising system confidentiality requirments.
- A user study that identified requirements for the design of security administration tools.

7.3 Future work

Ubiquitous computing covers a wide range of systems and applications, from ad hoc applications with very little infrastructure to equipment-intensive immersive virtual reality environments. This diversity in resource capability—power, network bandwidth and/or CPU—makes it difficult to generalize about such environments. This thesis research focuses on a subset of these environments—"smart" spaces, or physical spaces with the computing infrastructure to unify all the hardware into an interactive, programmable "Active Space". We have built an access control system for these environments and demonstrated its feasibility with a proof-of-concept implementation. In this section, we point out some related questions that were raised by this work, and are suitable for further research.

As computing environments get more decentralized, security policy configuration and administration becomes a problem. Ubiqutious computing environments need to compose policies obtained from different sources that have authority for the different components in the environment. This becomes particularly important in the area of privacy policies, which are important for ubiquitous computing. Automatically evaluating whether a space complies with a user's privacy policies, and vice versa, before starting applications, would be a useful property. While our work allows for policy composition in limited ways, a more general composition framework would be useful.

Access control depends on authentication and the proper use of credentials. In traditional systems, the management of these credentials is relatively straightforward. In ubiquituous computing systems, users may not have a terminal session associated with them, and associating credentials with users automatically is a challenge. In our environment, this is solved with the use of location-tracking and/or personal storage, but techniques for binding credentials to users in other environments need to be researched.

One of the problems with role-based access control is to identify which roles a user activates in a particular session. The principle of least privilege dictates that the "lowest" role be selected, but in practice, this often proves too restrictive, ending up with users activating all their roles or presenting all their credentials. While some of the work in trust negotiation deals with the question of presenting credentials in a way that user privacy is not compromised, role activation is still a problem in practical security systems.

Usability seems to be a major issue with the design of practical security systems; this is widely recognized today as one of the "grand challenges" for designing se-

curity systems [BLL+04]. More user studies to evaluate the usability of the security system would be useful.

The *Know* system investigates "useful" feedback; user studies to evaluate the usability of the feedback, and research on mechanisms for feedback in Active Spaces would be useful. We are exploring further avenues where feedback would be useful, as well as what privacy/confidentiality properties can be maintained while providing feedback. We use meta-policies for protecting the policy, and are in the process of identifying guidelines for the creation of such meta-policies and security properties that can be guaranteed.

Appendix A: ACS Interface Specification

```
/*
 * File: ACS.idl
 * Author: Geetanjali Sampemane, geta@cs.uiuc.edu
 * Description: Gaia Access Control Service interface description
 */

struct Credential {
  string name;
  string primary_role;
  sequence<string> roles;
  string addrString;
};

struct Permission {
  string method;
  sequence<string> roles; // Or maybe IP addresses
};

struct ServicePolicy {
  string name;
  sequence<string> Permission;
};

interface ACS {

  /**
   * Check if access is allowed
   *
   * @param owner   the owner of the layout file
   * @return        true/false if access is allowed/denied
   *                Various options control what happens in the case
   *                of undefined methods/services
```

```
 */
boolean isAllowed(in Credential cr, in string service,
                  in string method, in string addrString);

/**
 * Called when a user enters or leaves the space
 *
 * @param cr       User credential
 *
 * @return         Status true/false for success/failure
 */
boolean enterSpace(in Credential cr);
boolean leaveSpace(in Credential cr);

/**
 *
 * Mode change requests: can ask to switch to any of the
 * group modes -- ‘‘shared’’, ‘‘super’’ or ‘‘collab’’.  Assumes
 * that the proper authorization for super/collab has occurred.
 *
 *
 * @return         Status true/false for success/failure
 */
 boolean switchMode(in string target_mode)

/**
 *  Reads an Access List for a service from a file
 *
 *  @param service   Name of the service
 *  @param filename  Name of the file containing the new policy
 *
 *  @return true/false  Success/failure
 */
boolean readAList(in string service, in string filename);

/**
 *  Loads an new Access Control policy for a service.
 *
 *  @param service   Name of the service
```

```
     *  @param policy    Struct containing the policy
     *
     *  @return true/false  Success/failure
     */
    boolean readPolicy(in string service, in ServicePolicy policy);

    /**
     * Dumps the current AccessLists (of all services or a particular
     * service) in memory to stdout
     */
    void dumpSpace();
    void dumpAList(in string service);
};
```

Appendix B: Cognitive Walkthrough materials

These materials were used for a Cognitive Walkthrough conducted by Yong Liu, Geetanjali Sampemane and Kay Connelly. Analysts were given a fully-functional prototype of the administrative tool. They explored this prototype, walking through the steps required to complete tasks in each of the scenarios listed below, trying to identify usability problems. An online help document was implemented in the prototype and available for use.

B.1 List of scenarios

1. Add a new user to AS system (Goal A)
2. Add a new system role to AS system (Goal B)
3. Add a new application role to AS system (Goal C)
4. Add a new application to AS system (Goal D)
5. Add a new space to AS system (Goal E)
6. Map a user to specific system role (Goal F)
7. Assign a new permission to system role (Goal G)
8. Assign a new permission to application role (Goal H)
9. Install a new application to specific space (Goal I)
10. List current user-system role mappings in AS system (Goal J)
11. List permissions assigned to specific system role in AS system (Goal K)
12. List permissions assigned to specific application roles in AS system (Goal L)
13. List applications installed in specific space in AS system (Goal M)
14. Add a new space role to current space (Goal N)
15. Map a system role to specific space role (Goal O)
16. Map an application role to specific space role (Goal P)

17. List all role mappings (user-system role-space role-application role) in current space (Goal Q)
18. List all applications and corresponding methods in current space (Goal R)
19. List permissions assigned to specific space role in current space (Goal S)

B.2 Checklist provided to analysts

A checklist was provided to help the analysis and evaluation of the prototype. .

Examine in General:

1. Will the user try to achieve the right effect?

   The user may try to achieve the right effect:

   - by experience with this system or similar systems
   - if this is a new system replacing an old system, and the goal was part of the task on the previous system
   - because the system tells them, for example through a modal dialogue

2. What knowledge is needed to achieve the right effect? Will the user have this knowledge?

   To use a system requires knowledge of the task domain and the system supporting the task. There are four types of knowledge: procedural, conceptual, knowledge about the current system state and knowledge about the objects manipulated by the task.

   The user may have the knowledge

   - by experience with this system or similar systems
   - by experience with the task with other systems
   - because the system provides the necessary information

Examine the actions needed to satisfy the goal:

3 Will the user notice that the correct action is available?

   The user may notice that the correct action is available

   - by experience with this system or similar systems
   - if they see a representation of an action, for example a button

4 Will the user associate the correct action with the goal they are trying to achieve?

The user may make the association:

- by experience with this system or similar systems
- if there is some connection between the goal they are achieving and the action
- all other actions or choices look wrong

If the correct action is performed, examine the feedback:

5 Will the user perceive the feedback?

The user may perceive the feedback

- by experience with this system or similar systems
- if they are focusing on the area of the screen which provides the feedback
- if the feedback can not be ignored, for example audio

6 Will the user understand the feedback?

There is no guarantee that the user will understand the feedback if it is perceived. The user may understand the feedback

- by experience with this system or similar systems
- if the feedback is unambiguous

7 Will the user see that progress is being made towards solution of their task in relation to their goal?

The user may perceive progress

- by experience with this system or similar systems
- because they recognize a connection between the system response and their goal

Appendix C: Usability Test materials

The following are the materials used for conducting a usability test of an administrative tool for the Gaia access control system. The main purpose of the test is to predict the expected performance of an actual system (or space) administrator using the software and to remedy serious problems prior to deploying it. The usability test will measure the time to complete tasks and will identify errors and difficulties involved in using the prototype of the GUI. Simulated tasks include adding new roles, changing role-mappings and assigning new permissions to a role.

The specific questions that need to be answered:

1. Are all terms appearing on the GUI intuitive?
2. Are users able to recognize the tricky operations when adding a new application?
3. Are users able to recognize the key operation (add a new space role) when establishing new system-role to application-role mappings in a space?

C.1 Background questionnaire

Participants were asked to fill out a short questionnaire to gather background information.

Name:

Please answer the questions below in order to help us understand your background and experience.

1. Have you ever heard of Active Spaces?
2. Have you participated in the Gaia project?
3. If yes, please specify which component(s) you have worked with.
4. Do you know what Role Based Access Control is?

5. If yes, please specify how much you know about this model.
6. Are you familiar with the Gaia access control system?
7. If yes, specify how much you know about this sub-system of GAIA.

C.2 Orientation

Participants received a verbal introduction and orientation to the test, explaining the purpose and objective of the test and what was expected of them.

> We are here today to test how easy it is to use the Active Space RBAC system with a GUI. You will be performing some typical tasks with this GUI, and I'd like you to perform as you normally would. For example, try to work at the same speed and with the same attention to detail that you normally do. Do your best, but don't be all that concerned with results. This is a test of the Active Space Access Control system, which is still in prototype form, and it may not work as you expect. You may ask questions at any time, but I may not be able to answer some of them, since this is a study of the software and the information provided by it, and we need to see how it works with a person such as yourself working independently.
>
> During today's session, I will also be asking you to complete some forms and answer some questions. My only role here today is to discover both the flaws and advantages of this product from your perspective. So don't answer questions based on what you think I may want to hear. I need to know exactly what you think.
>
> While you are working, I will be sitting nearby taking some notes and timings. In addition, the session will be videotaped for the benefit of future analysis of this usability test.
>
> Do you have any questions?
>
> If not, then let us begin:

C.3 Terminology test

A terminology test was then administered, to evaluate whether the terms were understood before using the tool. Users were asked to define terms such as "user",

"system role", "space role" and other terms used in the administrative model.

C.4 Performance test

Participants were asked to carry out three tasks and were observed (and videotaped) while doing so.

MTC	Maximum Time to Complete
SCC	Successful Completion Criteria

Task Scenario A

Tom is a new comer to the Active Space system. He is a graduate student and will work in Computer Lab1. Tom needs a PDFViewer program to read papers in the lab, but now there is no such program in that room. As the system administrator, you need to make some configurations so that Tom can be signed up as a valid user and work with a PDFViewer in his lab. You will use "AS RBAC System Configuration Tool" to accomplish your task. Signal me when you feel you are done. Any questions before we begin?

Task Number:	A1
Task:	Add a new user "Tom".
SCC:	"Tom" appears in "Candidate User" list.
MTC:	1 min.

Task Number:	A2
Task:	Set user "Tom" to system role "student".
SCC:	"Tom" disappears from "Candidate User" list and appears in the "User" list with "Student" highlighted in "System Role"
MTC:	2 min.

Task Number:	A3
Task:	Add application "PDFViewer" to space "Computer Lab 1".
SCC:	"PDFViewer" disappears from the "Candidate Application" list and appears in the "Application" list, with "Computer Lab 1" highlighted in "Space".
MTC:	2 min.

Task Number:	A4
Task:	Assign the permission “PDFViewer:Read” to system role “Student”.
SCC:	“PDFViewer:Read” disappears from the “Candidate Permission” list and appears in the “Permission” list for “Student” in “System Role”.
MTC:	3 min.

Task Scenario B

Your department has bought a new application “MediaPlayer” and decided to use it in the Auditorium. As the system administrator, you need to install this application into the Active Space system. This application has two functions: “Play” (for speakers in the room to use) and “Watch” (for listeners in the room to use). Within the Auditorium, only the faculty members should be able to “Play” this application, but all the people in the department should be able to “Watch” it. Use the “AS RBAC System Configuration Tool” to make the necessary configurations and install “MediaPlayer” into the Auditorium. Signal me when you feel you are done. Any questions before we begin?

Task Number:	B1
Task:	Add a new application “MediaPlayer”.
SCC:	“MediaPlayer” appears in the “Candidate Application” list and all its functions appear in the “Candidate Permission” list.
MTC:	2 min.

Task Number	B2
Task:	Assign the permission “MediaPlayer.Play” to application role “Speaker”.
SCC:	“MediaPlayer.Play” disappears from the “Candidate Permission” list and appears in the “Permission” list
MTC:	2.0 min

Task Number	B3
Task:	Assign the permission “MediaPlayer.Watch” to application role “Listener”.
SCC:	“MediaPlayer.Watch” disappears from the “Candidate Permission” list and appears in the “Permission” list
MTC:	2.0 min

Task Number	B4
Task:	Add the application “MediaPlayer” to space “Auditorium”.
SCC:	“MediaPlayer” disappears from the “Candidate Application” list and appears in the “Application” list while “Auditorium” highlighted in “Space”.
MTC:	2.0 min

Task Number	B5
Task:	Add the permission “MediaPlayer.Play” to system role “Faculty”.
SCC:	“MediaPlayer.Play” disappears from the “Candidate Permission” list and appears in the “Permission” list while “Faculty” in “System Role”.
MTC:	2.0 min

Task Number	B6
Task:	Add the permission “MediaPlayer.Watch” to all system roles.
SCC:	“MediaPlayer.Watch” disappears from the “Candidate Permission” list and appears in the “Permission” list for each system role in “System Role”.
MTC:	3 min

Task Scenario C

Now suppose you are a space administrator in the Active Space RBAC system. In your space, you do not want a faculty member to be both a space role “listener” and a space role “speaker” Another rule is that, the space role “listener” in your space can only listen, and the space role “speaker” can only speak. But some faculty members need to both listen and speak in some circumstances. You need to make some configurations so as to satisfy those faculty members without breaking the rules. You are supposed to create only one new role in your space. After that, find out names of the people who can both listen and speak in your space now. You will use “AS RBAC Space Configuration Tool” to accomplish your task. Signal me when you feel you are done. Any questions before we begin?

Task Number	C1
Task:	Add a new space role “Whatever”.
SCC:	“Whatever” appears in the “Space Roles” list.
MTC:	3 min

Task Number	C2
Task:	Set Application role “Listener” to space role “Whatever”.
SCC:	“Listener” disappears from the “Candidate App Role” list and appears in the “Application Role” list, while “Whatever” highlighted in “Space Role”.
MTC:	2 min

Task Number	C3
Task:	Set application role “Speaker” to space role “Whatever”
SCC:	“Speaker disappears from the “Candidate App Role” list and appears in the “Application Role” list, while “Whatever” highlighted in “Space Role”.
MTC:	2 min

Task Number	C4
Task:	Set system role “Faculty to space role “Whatever”.
SCC:	“Faculty disappears from the “Candidate Sys Role” list and appears in the “System Role” list, while “Whatever” highlighted in “Space Role”.
MTC:	2 min

C.5 Post-test questionnaire

Particpants were asked to fill out a preference questionnaire pertaining to subjective perceptions of usability of the tool.

Please answer the following questions based on your experience using the Active Space RBAC GUI. Where appropriate, we would appreciate if you would explain your answers in the space provided below the questions.

1. Overall, I found the Active Space RBAC GUI easy to use. (Please check one)

   Strongly disagree

   Disagree

   Neither agree nor disagree

   Agree

   Strongly agree

2. I found the following aspects of the Active Space RBAC GUI particularly easy to use. (Please list from 0-3 aspects.)

   A

   B

   C

3. I found the following aspects of the Active Space RBAC GUI particularly difficult to use. (Please list from 0-3 aspects.)

   A

   B

   C

4. I found the terminology of the Active Space RBAC GUI clear and precise.

   Strongly disagree

Disagree

Neither agree nor disagree

Agree

Strongly agree

5. Using the following rating sheet, please circle the number nearest the term that most closely matches your feeling about the Active Space RBAC GUI.

Simple	1	2	3	4	5	Complex
Easy-to-use	1	2	3	4	5	Difficult-to-use
Friendly	1	2	3	4	5	Unfriendly
Professional	1	2	3	4	5	Unprofessional
I like	1	2	3	4	5	I dislike

6. Please add any comments in the space provided that you feel will help us to evaluate the Active Space RBAC GUI.

C.6 Evaluation measures

The following evaluation measures were collected and calculated:

- The average time to complete each task and each scenario across all participants.
- The percentage of participants who finished each task successfully versus those who had errors from which they could not recover.
- Error classification: each error was classified and the source of each class of errors indicated.
- Percentage of participants who gave the correct definition of the terminology test one.
- Percentage of participants who gave the correct definition of the terminology test two.
- Comparison of terminology test one and two scores for each definition.
- Participant rankings of usability of the administrative tool.

References

[AHR] The Aware Home research initiative. `http://www.awarehome.gatech.edu`.

[AMRCM03] Jalal Al-Muhtadi, Anand Ranganathan, Roy Campbell, and M. Dennis Mickunas. Cerberus: A context-aware security scheme for smart spaces. In *Proceedings of the first IEEE Annual Conference on Pervasive Computing and Communications (PerCom)*, pages 489–496, Forth Worth, TX, March 2003.

[AS00] Gail-Joon Ahn and Ravi Sandhu. Role-based authorization constraints specification. *ACM Trans. Inf. Syst. Secur.*, 3(4):207–226, 2000.

[BB94] Adrian Bullock and Steve Benford. An approach to access control for collaborative virtual environments. In *Proc. of the 6th ERCIM Workshop*, pages 233–264, Stockholm, 1994.

[BBF01] Elisa Bertino, Piero Andrea Bonatti, and Elena Ferrari. TRBAC: A temporal role-based access control model. *ACM Trans. Inf. Syst. Secur.*, 4(3):191–233, 2001.

[BdS00] Piero Bonatti, Sabrina de Capitani di Vimercati, and Pierangela Samarati. A modular approach to composing access control policies. In *Proceedings of the 7th ACM Conference on Computer and Communications Security (CCS)*, pages 164–173. ACM Press, 2000.

[BDS01] Piero Bonatti, Ernesto Damiani, and Pierangela Samarati. A component-based architecture for secure data publication. In *Proceedings of 17th Annual Computer Security Applications Conference (ACSAC)*, pages 309–318, New Orleans, LA, December 2001.

[BDS02] Piero Bonatti, Sabrina De Capitani di Vimercati, and Pierangela Samarati. An algebra for coomposing access control policies. *ACM Transactions on Information and System Security (TISSEC)*, 5(1):1–35, 2002.

[BFA99] Elisa Bertino, Elena Ferrari, and Vijay Atluri. The specification and enforcement of authorization constraints in workflow management systems. *ACM Trans. Inf. Syst. Secur.*, 2(1):65–104, 1999.

[BG99] Michael Beigl and Hans-Werner Gellersen. Ambient telepresence. In *Proceedings of the Workshop on Changing Places*, pages 63–69, London, UK, 1999.

[BLL+04] Steve Bellovin, Carl Landwehr, Jean-Claude Laprie, Roy Maxion, and Ravi Iyer (mod.). Panel: Grand challenges and open questions in trusted systems. ITI Workshop on Dependability and Scalability, Urbana, IL, December 2004. `http://www.iti.uiuc.edu/seminars/Dec2004_ITI_Workshop.html`.

[Bry86] R.E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, 1986.

[BS02] Piero A. Bonatti and Pierangela Samarati. A uniform framework for regulating service access and information release on the Web. *Journal of Computer Security*, 10(3):241–271, 2002.

[CFZA02] Michael J. Covington, Prahlad Fogla, Zhiyuan Zhan, and Mustaque Ahamad. A context-aware security architecture for emerging applications. In *Proceedings of 18th Annual Computer Security Applications Conference (ACSAC)*, pages 249–258, Las Vegas, NV, December 2002. IEEE.

[CHRC01] Renato Cerqueira, Christopher K. Hess, Manuel Román, and Roy H. Campbell. Gaia: A development infrastructure for Active Spaces. In *Workshop on Application Models and Programming Tools for Ubiquitous Computing (held in conjunction with the UBICOMP 2001)*, Atlanta, GA, September 2001.

[CLS+01] Michael J. Covington, Wende Long, Srividhya Srinivasan, Anind K. Dev, Mustaque Ahamad, and Gregory D. Abowd. Securing context-aware applications using environment roles. In *Proceedings of the 6th ACM symposium on Access control models and technologies (SACMAT'01)*, pages 10–20. ACM Press, 2001.

[CMA00] Michael J. Covington, Matthew J. Moyer, and Mushtaque Ahamad. Generalized role-based access control for securing future applications. In *23rd National Information Systems Security Conference*, pages 115–125, Baltimore, MD, October 2000. National Institute of Standards and Technology, National Computer Security Center.

[CMM72] R. W. Conway, W. L. Maxwell, and H. L. Morgan. On the implementation of security measures in information systems. *Commun. ACM*, 15(4):211–220, 1972.

[CTWS02] Eve Cohen, Roshan K. Thomas, Willian Winsborough, and Deborah Shands. Models for coalition-based access control (CBAC). In *Proceedings of the seventh ACM Symposium on Access Control models and technologies (SACMAT)*, pages 97–105. ACM, June 2002.

[DA00] Anind Dey and Gregory Abowd. The context toolkit: Aiding the development of context-aware applications. In *Workshop on Software Engineering for Wearable and Pervasive Computing*, Limerick, Ireland, June 2000.

[Den82] Dorothy Denning. *Cryptography and Data Security*. Addison-Wesley Publishing Company, 1982.

[DHI78] W. Dzida, S. Herda, and D. Itzefeldt. User-perceived quality of interactive systems. *IEEE Trans. Software Eng.*, SE-4(4):270–276, July 1978.

[Dij75] Edsger W. Dijkstra. Guarded commands, nondeterminacy, and formal derivation of programs. *Communications of the ACM*, 18(8):453–457, August 1975.

[DSA99] Anind K. Dey, Daniel Salber, and Gregory D. Abowd. A context-based infrastructure for smart environments. In *Proceedings of the 1st International Workshop on Managing Interactions in Smart Environments (MANSE'99)*, Dublin, Ireland, December 1999.

[Epp94] David Eppstein. Finding the k shortest paths. In *Proc. 35th Symp. Foundations of Computer Science*, pages 154–165. IEEE, November 1994.

[FK92] David F. Ferraiolo and D. Richard Kuhn. Role-based access controls. In *Proceedings of the 15th NIST-NCSC National Computer Security Conference*, pages 554–563, Baltimore, MD, October 1992.

[FKT01] Ian Foster, Carl Kesselman, and Steven Tuecke. The anatomy of the Grid: Enabling scalable virtual organizations. *International Journal of Supercomputer Applications*, 15(3):200–222, 2001.

[FPP+02] Eric Freudenthal, Tracy Pesin, Lawrence Port, Edward Keenan, and Vijay Karamcheti. dRBAC: Distributed role-based access control for dynamic coalition environments. In *Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS'02)*, pages 411–420, Vienna, Austria, jul 2002. IEEE.

[Gai00] Gaia: Active Spaces for ubiquitous computing. `http://gaia.cs.uiuc.edu`, 2000.

[GD72] G. Graham and Peter Denning. Protection—principles and practice. In *Spring Joint Computer Conference—AFIPS Conference Proceedings*, volume 40, pages 417–429, 1972.

[Gil01] Binny Sher Gill. Dynamic policy-driven role-based access control for active spaces. Master's thesis, University of Illinois at Urbana-Champaign, 2001.

[GKK+02] Virgil D. Gligor, Himanshu Khurana, Radostina K. Koleva, Vijay G. Bharadwaj, and John S. Baras. On the negotiation of access control policies. In *Revised Papers from the 9th International Workshop on Security Protocols*, pages 188–201. Springer-Verlag, 2002.

[GMPT01] Christos K. Georgiadis, Ioannis Mavridis, George Pangalos, and Roshan K. Thomas. Flexible team-based access control using contexts. In *Proceedings of SACMAT 01*, Virginia, USA, May 2001. ACM.

[Gra] Jonathan Graehl. `kbest`, a C++ library for efficiently finding the k shortest paths in a graph. Available from `http://jonathan.graehl.org/kbest.zip`.

[GSSS02] D Garlan, D Siewiorek, A Smailagic, and P Steenkiste. Project Aura: Toward distraction-free pervasive computing. *IEEE Pervasive Computing*, pages 22–31, April–June 2002.

[HB00] Felix Hupfeld and Michael Beigl. Spatially aware local communication in the RAUM system. In *Proceedings of the IDMS*, pages 285–296, Enschede, Niederlande, Oct 2000.

[HC03] Christopher K. Hess and Roy H. Campbell. A context-aware data management system for ubiquitous computing applications. In *Proceedings of the 23rd International Conference on Distributed Computing Systems (ICDCS2003)*, pages 294–301, Providence, RI, May 2003.

[Hes03] Christopher K. Hess. *The Design and Implementation of a Context-Aware File System for Ubiquitous Computing Applications*. PhD thesis, University of Illinois at Urbana-Champaign, 2003.

[HRU76] Michael A. Harrison, Walter L. Ruzzo, and Jeffrey D. Ullman. Protection in operating systems. *Commun. ACM*, 19(8):461–471, 1976.

[HS04] Urs Hengartner and Peter Steenkiste. Implementing access control to people location information. In *Proceedings of the 9th ACM Symposium on Access Control Models and Technologies*, pages 11–20, Yorktown Heights, NY, USA, June 2004. ACM Press.

[JFW02] Brad Johanson, Armando Fox, and Terry Winograd. The Interactive Workspaces project: Experiences with ubiquitous computing environments. *IEEE Pervasive Computing magazine*, 1(2):67–74, April–June 2002.

[JMWU91] Robin Jeffries, James R. Miller, Cathleen Wharton, and Kathy Uyeda. User interface evaluation in the real world: a comparison of four techniques. In *CHI '91: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 119–124. ACM Press, 1991.

[JP96] Trent Jaeger and Atul Prakash. Requirements of role-based access control for collaborative systems. In *Proceedings of the ACM RBAC Workshop*, 1996.

[KCF92] Claire-Marie Karat, Robert Campbell, and Tarra Fiegel. Comparison of empirical testing and walkthrough methods in user interface evaluation. In *CHI '92: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 397–404. ACM Press, 1992.

[KFJ01] Lalana Kagal, Tim Finin, and Anupam Joshi. Trust-based security in pervasive computing environments. *IEEE Computer*, December 2001.

[KGL02] Himanshu Khurana, Virgil Gligor, and John Linn. Reasoning about joint administration of access policies for coalition resources. In *Proceedings of the 22nd IEEE International Conference on Distributed Computing Systems (ICDCS)*, pages 429–440, Vienna, Austria, July 2002.

[KHJ03] Håkan Kvarnström, Hans Hedbom, and Erland Jonsson. Protecting security policies in ubiquitous environments using one-way functions. In D.Hutter et al., editors, *Security in Pervasive Computing 2003*, volume 2802 of *LNCS*, pages 71–85. Springer-Verlag, Heidelberg, 2003.

[KKC02] Arun Kumar, Neeran Karnik, and Girish Chafle. Context sensitivity in role-based access control. *ACM SIGOPS Operating Systems Review*, 36(3):53–66, 2002.

[Kle90] Daniel V. Klein. "foiling the cracker"—A survey of, and improvements to, password security. In *Proceedings of the second USENIX Workshop on Security*, pages 5–14, Summer 1990.

[KP88] Glenn E. Krasner and Stephen T. Pope. A cookbook for using the model view controller user interface paradigm in smalltalk-80. *Journal of Object-Oriented Programming*, 1(3):26–49, August–September 1988.

[KPF01] Myong H. Kang, Joon S. Park, and Judith N. Froscher. Access control mechanisms for inter-organizational workflow. In *Proceedings of the 6th ACM Symposium on Access Control Models and Technologies*, pages 66–74, Virginia, USA, May 2001. ACM.

[KSC04] Apu Kapadia, Geetanjali Sampemane, and Roy H. Campbell. KNOW why your access was denied: regulating feedback for usable security. In *Proceedings of the 11th ACM conference on Computer and Communications Security (CCS '04)*, pages 52–61. ACM Press, 2004.

[Lam71] Butler W. Lampson. Protection. In *Proceedings of the 5th Princeton Conference on Information Sciences and Systems*, pages 437–443, Princeton, NJ, March 1971. Princeton University. reprinted in Operating Systems Review, 8, 1, Jan 1974, pp.18–24.

[Lan01] Marc Langheinrich. Privacy by design — principles of privacy-aware ubiquitous systems. In *Proceedings of Ubicomp 2001*, Atlanta, GA, September 30 – October 2 2001.

[LN99] J. Lind-Nielsen. BuDDy – a binary decision diagram package. Technical Report IT-TR: 1999-028, Technical University of Denmark, 1999.

[LRP95] John Lamping, Ramana Rao, and Peter Pirolli. A focus+context technique based on hyperbolic geometry for visualizing large hierarchies. In *Proceedings of the ACM SIGCHI conference on human factors in computing systems*, Denver, May 1995.

[LSC03] Yong Liu, Geetanjali Sampemane, and Kay Connelly. Experiences using usability techniques in development of Active Spaces RBAC GUI. In *Proceedings of forUSE 2003, Second International Conference on Usage-Centered Design*, pages 37–44, October 2003.

[NL94] J Nielsen and J Levy. Measuring usability: preference vs. performance. *Communications of the ACM*, 37(4):66–75, 1994.

[NM94] Jakob Nielsen and Robert L. Mack, editors. *Usability Inspection Methods*. John Wiley and Sons, 1994.

[NO95] Matunda Nyanchama and Sylvia L. Osborn. Modeling mandatory access control in role-based security systems. In *Proceedings of IFIP Workshop on Database Security*, pages 129–144, 1995.

[OMG] Common object request broker architecture (CORBA/IIOP). The Object Management Group.

[Orb] Orbacus ORB. `http://www.orbacus.com`.

[OSM00] Sylvia Osborn, Ravi Sandhu, and Qamar Munawer. Configuring role-based access control to enforce mandatory and discretionary access control policies. *ACM Trans. Inf. Syst. Secur.*, 3(2):85–106, 2000.

[Oxy] MIT project Oxygen. http://oxygen.lcs.mit.edu.

[Pen96] Alex P. Pentland. Smart rooms. *Scientific American*, April 1996.

[PM01] S. Pettifer and J. Marsh. A collaborative access model for shared virtual environments. In *Proceedings of IEEE WETICE 01*, pages 257–272. IEEE Computer Society, June 2001.

[PTD02] Charles E. Phillips, Jr., T.C. Ting, and Steven A. Demurjian. Information sharing and security in dynamic coalitions. In *Proceedings of the 7th ACM Symposium on Access Control Models and Technologies*, pages 87–96, Monterey, CA, June 2002. ACM.

[RC03] Manuel Román and Roy H. Campbell. Providing middleware support for active space applications. In *Proceedings of ACM/IFIP/USENIX International Middleware Conference (Middleware 2003)*, Rio de Janeiro, Brazil, 2003.

[RHC+02] Manuel Román, Christopher K. Hess, Renato Cerqueira, Anand Ranganathan, Roy H. Campbell, and Klara Nahrstedt. GaiaOS: A middleware infrastructure to enable Active Spaces. *IEEE Pervasive Computing*, pages 74–83, October–December 2002.

[Rii00] Sirpa Riihiaho. *Experiences with usability evaluation methods*. Licentiate's thesis, Helsinki University of Technology, 2000. Available from `http://www.soberit.hut.fi/~sri/Riihiaho_thesis.pdf`.

[Rom03] Manuel Roman. *An Application Framework for Active Space Applications*. PhD thesis, University of Illinois at Urbana-Champaign, May 2003.

[Rub94] Jeffrey Rubin. *Handbook of Usability Testing: How to plan, design and conduct effective tests*. John Wiley and sons, New York, 1994.

[SB02] Narendra Shankar and Dirk Balfanz. Enabling secure ad-hoc communication using context-aware security services. Workshop on Security in Ubiquitous Computing, Ubicomp 2002, September 2002.

[SBM99] Ravi Sandhu, Venkata Bhamidipati, and Qamar Munawer. The ARBAC97 model for role-based administration of roles. *ACM transactions on Information and System Security*, pages 105–135, February 1999.

[SCFY96] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinsten, and Charles E. Youman. Role-based access control models. *IEEE Computer*, 20(2):38–47, 1996.

[Sch00] Fred Schneider. Enforceable security policies. *ACM Transactions on Information and System Security*, 3(1):30–50, 2000.

[SD92] HongHai Shen and Prasun Dewan. Access control for collaborative environments. In John Turner and Robert Kraut, editors, *Proceedings of ACM Conference on Computer-Supported Collaborative Work (CSCW)*, pages 51–58. ACM Press, 1992.

[SS75] Jerome H. Saltzer and Michael D. Schroeder. The protection of information in computer systems. In *Proceedings of the IEEE*, volume 63, pages 1278–1308, September 1975.

[TS97] Roshan K. Thomas and Ravi S. Sandhu. Task-based authorization controls (TBAC): A family of models for active and enterprise-oriented authorization management. In *Proceedings of the IFIP WG11.3 Workshop on Database Security*, pages 166–181, Lake Tahoe, CA, August 1997. Chapman and Hall.

[UIC] Universally interoperable core. `http://www.ubi-core.com`.

[Vis01] Prashant Viswanathan. Security architecture in Gaia. Master's thesis, University of Illinois at Urbana-Champaign, 2001.

[Wei91] Mark Weiser. The computer for the 21st century. *Scientific American*, pages 94–104, September 1991.

[WJ99] Alma Whitten and J.D.Tygar. Why Johnny can't encrypt: A usability evaluation of PGP 5.0. In *Proceedings of the 8th USENIX Security Symposium'99*, Washington, D.C., August 1999.

[WKJ+01] Patrik Werle, Fredrik Kilander, Martin Jonsson, Peter Lönnqvist, and Carl Gustaf Jansson. A ubiquitous service environment with active documents for teamwork support. In *Proceedings of Ubicomp 2001*, Atlanta, GA, September 30–October 2 2001.

[WL04] William H. Winsborough and Ninghui Li. Safety in automated trust negotiation. In *Proceedings of the 2004 IEEE Symposium on Security and Privacy*, pages 147–160, Oakland, CA, May 2004. IEEE Press.

[WRLP94] Cathleen Wharton, John Rieman, Clayton Lewis, and Peter Polson. *The cognitive walkthrough method: a practitioner's guide*, pages 105–140. John Wiley & Sons, Inc., 1994.

[WWB03] Andy Ward, Paul Webster, and Peter Batty. Local positioning systems—technology overview and applications. White paper available from `http://www.ubisense.net`, 2003.

[Yee02] Ka-Ping Yee. User interaction design for secure systems. In *Proceedings of the 4th International Conference on Information and Communications Security*, pages 278–290. Springer-Verlag, 2002.

[Yee03] Ka-Ping Yee. Secure interaction design and the principle of least authority. In *Workshop on Human-Computer Interaction and Security Systems, part of CHI2003*, Fort Lauderdale, FL, apr 2003.

[Yee04] Ka-Ping Yee. Aligning usability and security. *IEEE Security and Privacy Magazine*, pages 48–55, September 2004.

[YWS03] Ting Yu, Marianne Winslett, and Kent E. Seamons. Supporting structured credentials and sensitive policies through interoperable strategies for automated trust negotiation. *ACM Trans. Inf. Syst. Secur.*, 6(1):1–42, 2003.

[ZP04] G. Zhang and M. Parashar. Context-aware dynamic access control for pervasive applications. In *Proceedings of the Communication Networks and Distributed Systems Modeling and Simulation Conference (CNDS 2004), Western MultiConference (WMC)*, San Diego, CA, January 2004.

[ZS96] Mary Ellen Zurko and Richard T. Simon. User-centered security. In *Proceedings of the Workshop on New Security Paradigms (NSPW)*, pages 27–33, Lake Arrowhead, CA, September 1996.

Author's Biography

Geetanjali Sampemane was born in Bombay, India on 2 March, 1970. She received a B.Tech degree in Mechanical Engineering from the Indian Institute of Technology Bombay in 1991, an M.S. degree in Computer Science from the University of Illinois at Urbana-Champaign in 2001 and a Ph.D. in Computer Science from the University of Illinois at Urbana-Champaign in 2005.

Geetanjali worked on the Educational and Research Networking (ERNET) project at the National Centre for Software Technology in Bombay, India during 1991-1995. She worked as a Networking Specialist for the Sustainable Development Networking Programme of the United Nations Development Programme from 1995 to 1997 before starting graduate work in Computer Science at the University of Illinois at Urbana-Champaign.

Geetanjali worked on the High Performance Virtual Machines (HPVM) project, designing and implementing a performance monitoring system for the high-performance cluster of workstations, to obtain her Master's degree in 2001. She then switched to working on security issues for ubiquitous computing, implementing an access control system for the Gaia project. Her research interests include system and network design, security, usability and performance evaluation.