Access Control for Active Spaces[*]

Geetanjali Sampemane geta@cs.uiuc.edu
Prasad Naldurg naldurg@cs.uiuc.edu
Roy H. Campbell rhc@cs.uiuc.edu

Department of Computer Science
University of Illinois at Urbana-Champaign

Abstract

Active Spaces are physical spaces augmented with heterogeneous computing and communication devices along with supporting software infrastructure. This integration facilitates collaboration between users, and promotes greater levels of interaction between users and devices. An Active Space can be configured for different types of applications at different times. We present an access control system that automates the creation and enforcement of access control policies for different configurations of an Active Space. Our system explicitly recognizes different modes of cooperation between groups of users, and the dependence between physical and virtual aspects of security in Active Spaces.

Our model provides support for both discretionary and mandatory access control policies, and uses role-based access control techniques for easy administration of users and permissions. We dynamically assign permissions to user roles based on context information. With the help of an example scenario, we show how we can create dynamic protection domains. This allows administrators and application developers the ability to customize access control policies on a need-to-protect basis. We also provide a semi-formal specification and analysis of our model and show how we preserve safety properties in spite of dynamic changes to access control permissions. We also show how our model preserves the principle of least privilege, promotes separation of duty, and prevents rights-amplification.

1. Introduction

An Active Space is an interactive environment consisting of physical spaces—such as offices and meeting rooms—integrated with embedded computing and communication hardware and software. This integration enables interactive information exchange between users and the space, allowing new types of collaborative applications, as well as augmenting traditional models of human–computer interaction. An example of an Active Space is a "smart room" with multimedia devices, and a variety of desktop, laptop, and hand-held computers belonging to different users, who may or may not be physically in the room. Such rooms are becoming common in universities and workplaces, and are used for activities such as meetings and lectures.

The software infrastructure for Active Spaces provides a programmable interface to various multimedia mechanisms for input, output, and control of the physical and virtual aspects of the space. Software services, such as location tracking, data storage, dynamic resource discovery and others, abstract common functionality and simplify the development of "active" applications. In our prototype, these services are implemented in middleware as a meta operating system called Gaia [16], that runs on top of the native system software on the computers and devices in the space.

Using these services, an Active Space may be configured to provide support to different activities at different points in time. For example, the same room may be used as an intelligent meeting room and later change into an intelligent lecture room. Each activity-specific incarnation of an Active Space is called a "Virtual Space". While much research has gone towards the design of software [16, 13] and applications [21] for Active Spaces, security issues have not yet been explored in detail.

One of the main security concerns for Active Spaces is access control. Access control policies and mechanisms are necessary to ensure that users only use the resources (both hardware and software) in an Active Space in authorized ways, and to allow shared use of the space. Smart rooms are typically shared by different groups of users for different activities at different times. These different virtual spaces have varying access control requirements, and the access control policies and mechanisms should allow seamless transitions between virtual spaces, without sacrificing security properties. In addition, the policies should be easy to configure, enforce, and administer.

We contend that any access control model for Active Spaces must provide dynamic support for explicit cooperation between different users, groups of users, and devices. The model should also ensure that an individual's access rights do not "leak" across groups, and users neither gain (amplify) nor lose (attenuate) their rightful privileges during collaboration. In addition to access control in the

[*]This research is supported by the National Science Foundation, NSF 0086094, NSF 98-70736 and NSF 99-72884 CISE.

virtual world, the nature of the environment allows physical aspects of the real world to affect security properties of the space. For example, a security policy may prohibit a group of users access to certain files, but these users may be able to circumvent the policy by looking at the files on the display monitors of other authorized users in the room. Designing an access control system for such a heterogeneous, dynamic environment is a significant challenge.

In this paper, we describe a novel access control model that supports customizable access control policies, and integrates physical and virtual aspects of the environment into the access control decision mechanisms. Our prototype access control system can reconfigure an Active Space dynamically to support different access control policies for different virtual spaces, depending on the set of users and the activity being undertaken in the space.

To implement our access control model, we use a form of role-based access control (RBAC)[8, 17], which can support both mandatory and discretionary access control policies. Our system recognizes three kinds of user roles: system roles, space roles and application roles. *System roles* are assigned when user accounts are created, and define users' generic permissions for certain classes of objects (or resources) within the entire system. A system typically corresponds to an administrative domain.

Within each Active Space, however, access control policies are expressed in terms of *space roles*. Each Active Space within a system has a space administrator who sets access control policies for resources within that space. When users enter a space, their system role is mapped into an appropriate space role automatically. Our system also allows *application roles* that allow an application to specify a customizable access control policy. For example, a presentation application may require that only the "presenter" role be allowed to control the slides in the presentation. Application roles are mapped into space roles and access control is performed on these resulting space roles. The space administrator creates the mapping between system and application roles and space roles by examining the generic permissions in the system role and creating instances of permissions that refer to real resources in the space. This hierarchical separation of duty simplifies implementation and makes our design scalable. The mappings are designed to ensure that users are never allowed more permissions than their system role permits.

We also build an explicit notion of cooperation into our access control model using the concept of the *space mode*. We define four distinct modes of collaboration in our model: individual, shared, supervised-use and collaborative. The mode of the space depends on the users within the space and the activity being performed. *Individual* mode allows a single user in a space all the rights that are given by her role. The space switches into *shared* mode when there are more users in it. By default, in the shared mode, the space only allows actions that *all* users currently in the space are allowed, thus ensuring that safety properties are maintained. This is similar to conventional low-water mark, high-water mark approaches [3]. However, this may prove too restrictive for some applications. To overcome this, the room can go into *supervised-use* shared mode, where a "supervisor" may be allowed to perform certain actions that would be disallowed in group-use mode.

In addition to these, we also introduce the concept of a cooperative mode where each user is given all the permissions of all the users in the space. This *collaborative* mode is useful when a group of mutually trusting users get together to complete a task that would not be possible for any of them individually. This mode has to be used with care, as rights-amplification may have undesirable side effects. We analyze and validate the safety properties of the different modes of access with the help of a semi-formal specification of our prototype access control system.

The rest of the paper is organized as follows: Section 2 describes the system architecture and Section 3 discusses the implementation of the access control system with an illustrative example. Section 4 has a semi-formal model and validation of the access control system, Section 5 discusses related work, and finally Section 6 concludes.

2. Architecture

In this section, we describe the design of our access control system. We first describe the system components, and then the management of users and access control policies.

One of the prerequisites for access control is a method for authenticating users. In our system, an *Authentication Service* reliably identifies the user and issues her a *credential*. This credential accompanies every request for service that a user (or an application on behalf of a user) makes in the space, and is used by the *Access Control Service* to identify the requester. The specific authentication mechanisms are not the focus of this paper; we can use any authentication mechanism that securely transmits the credential to the user, in such a manner that credentials cannot be tampered with, stolen, or forged. Given these properties, the access control system can rely on the credential to identify the requester.

We assume that physical access to the space is restricted to authorized users. A typical implementation of this is by requiring users to swipe access cards at an electronic doorlock to let them in[1]. This method is obviously vulnerable to users sneaking in with other valid users, but a user who manages to enter the space without a valid credential cannot access any resources. Also, other users in the space can detect intruders, so this is a reasonable mechanism to build upon. The Active Space thus knows who the users in a space are, and uses this information to

[1]Visitors could be assigned temporary cards to identify them as such.

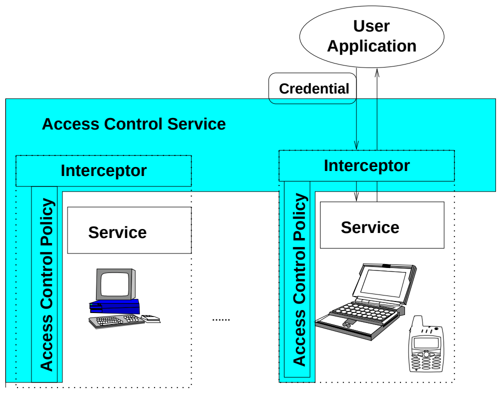

Figure 1. Access Control System Architecture

configure the security policies.

2.1. System description

Figure 1 shows the components of our access control system. Applications run on behalf of *users* and make use of *devices* in the space. Software *services* provide interfaces for user applications to use the devices. The space administrator sets access control policies, which are in the form of access control lists for the services in the space.

The access control service includes interceptors that intercept all requests to services, and only permit authorized ones. The credential accompanying the request is used to identify the requester, and permissions for the operation are checked on the access list for the service in the space. If the requester has sufficient permissions for the operation attempted, it is allowed to proceed. The interceptor may be implemented by the service, or by the space software infrastructure, transparent to applications and users.

The main objective of the access control system is to control access to the services within a space. Applications use services to access devices. Access control for devices thus involves physical access control to the devices, as well as software access control to the services that enable applications to use the devices.

The access control system works as follows:

- The space administrator sets up access control policies for all services in the space. All access to devices is through services.
- Users enter the Active Space after authenticating themselves. Physical access to the space is restricted to legitimate users. The space knows who its current occupants are, and switches to the appropriate mode.
- 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.

In the following section, we describe how the security policies are set.

2.2. Users, Roles and Policies

We use a form of role-based access control (RBAC) to administer security policies within a space. RBAC simplifies 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 allocate permissions to these roles.

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*. 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.

A system may contain one or more Active Spaces within it. We introduce the concept of a *space role*, with associated *space permissions*, which are just permissions to resources within the space. Each Active Space has a *space administrator*, whose task is to setup access control policies to resources in 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 valid within A.

A user’s permissions within a space cannot exceed 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.

We use the notion of *virtual spaces* to denote an Active Space configured for a particular activity. For example, an Active Space could be used to conduct a *meeting*, which would be a virtual space, in which the only space roles allowed to be present are *participant* and *chair*. The same Active Space could later be used to conduct a *lecture*, i.e. turn into a different virtual space, and the valid space roles could be *speaker* and *listener*.

Application roles are used to specify access control policies for applications. These application roles are then mapped into space roles by the space administrator. An 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 seminar `speaker` must have access to a projection device and `listeners` 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 an application can run in a space, he or she also creates the mappings from its 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 enforced only in terms of these space roles.

Each service in the space is associated with an access control policy, set by the space administrator. This policy is in the form of an *access list*, containing an entry for each space role and listing its permissions for this service. The access control decision involves looking up the requester's role in this access list, and verifying that it is allowed to perform the attempted operation. Since this is a simple table lookup, it can be performed quickly. The access control policy of a service can be changed by modifying the access list.

This model explains the working of the Access Control System for a single user in a space. However, as mentioned earlier, the security policy for a space is affected by the users in the space. This is described next.

2.3. Modes of Access

Unlike in traditional multi-user operating systems, users cannot be completely isolated from each other in an Active Space. Therefore, the security policy of a smart room is affected by the users in it. A user's permissions may be restricted when other users enter the space. For example, a user may be allowed to play loud music only when alone in an Active Space.

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. The Active Space allows three modes of access:

1. 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.

2. *Group modes*, with multiple users in the space, are the most typical usage mode. The Access Control System grants every member of the group the same permissions. In the simplest case, a group of users *shares* the space without 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.

   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, we point out that 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.

   The space switches automatically from *individual* to a *group* mode (after notifying any user in it) when a second user enters it. The collaborative mode is only entered if explicitly requested.

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. If P_m is the set of permissions allowed to a user in mode m, we assert that $P_{shared} \subseteq P_{supervisor} \subseteq P_{individual}$, for all users in the space, thereby preserving the principle of tranquility. When an individual (authorized) user requests supervisor privileges, the space switches to *supervised* 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 modes of access are similar to protection domains

Table 1. Mode Switching in an Active Space

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

used by traditional operating systems. These group domains are automatically created when a group of users is present in an Active Space, and Table 1 shows the allowed domain switches in our system. Traditional operating systems like UNIX set a protection domain per user or per process, independent of other users of the machine. In contrast, the mode of an Active Space is dependent on the set of users in the space, and an additional user entering/leaving can cause a domain 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.

2.4. Policy Management

The management of our access control system is distributed between the system administrator and the space administrator. The system administrator's task consists of assigning new users to appropriate system roles, which 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.

We argue that having a space administrator who controls access to each space is a good model for most real-world environments, because it exploits the *separation of duty* principle. 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, and form a natural grouping of permissions that need to be set.

3. Implementation

We now describe our prototype implementation of the access control system, and, with the help of an example, the reconfiguration of the space access control policies.

3.1. Sessions

To allow dynamic reconfiguration of access control policies, we introduce the concept of an Active Space *session*. Each session contains a mapping from system and application roles to space roles for users currently in the space. 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. 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.

3.2. Prototype implementation

We are building a prototype implementation of this access control system for an Active Space. Our Active Space system, Gaia, is implemented in middleware, in the form of CORBA services running on the native operating system. Gaia provides a set of "kernel services" such as naming, events and context, to allow applications to use the resources in the space effectively. Devices within the space present a CORBA interface to applications.

In this environment, the natural choice for intercepting requests and enforcing access control policies is to use CORBA Portable Interceptors[10]. 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 space administrator can install the hooks to implement the space access control policy when a service or application is installed in an Active Space.

Each service in the Active Space has an access list. The space also maintains the "current role translation" which maps the system roles of every user present to a valid space role. This mapping preserves the safety property that it will not increase a user's permissions.

Client-side interceptors attach user credentials to requests for services, and server-side interceptors use these credentials to perform access control. If the request contains missing or invalid credentials, the request fails.

The credential contains the user's system roles, which are mapped to a space role in a session. A user may also belong to an application role during an application session—this application role is similarly mapped into a space role. The access list for the service consists of a list of space roles, and the permissions they are allowed. These mappings do not change during the lifetime of a session, and may be cached to improve performance.

Since CORBA Interceptors intercept *all* orb-mediated communication, it is important to reduce the performance penalty of the interception. In our system, the access control decision implemented in the interceptors is in the form of a simple table lookup. The access control enforcement verifies that the *current* space role of the requester is allowed to perform the action being requested. If the access control fails, a CORBA exception is thrown. The performance overhead is negligible for typical Active Space applications.

When the space is reconfigured for a different application, the only parts of the access control system that need changing are the current role translation table and (perhaps) the access lists of the services in the space. The current role translation table should be updated to contain the new role mappings. The access lists for the services has to be updated to contain the new permissions for the *group* modes, when the composition of the group changes.

3.3. Example application

We now proceed to describe the implementation and operation of the access control system by working through a simple example scenario for a smart room. For each situation, we describe the actions taken by various components of the access control system.

Consider a system with two types of users, students and faculty. This system contains an Active Space AS_1, which is a "smart room" containing a variety of devices. There are four system roles: student, faculty, sysadm, spaceadm. Let us consider just two of the devices in the smart room, a projector P and a whiteboard B.

The space administrator wishes to install a *lecture* application in the space. This application has two roles—*speaker* and *listener*. The space administrator creates two space roles and assigns them the necessary permissions to run the application. The permissions are shown in the access matrix in Table 2.

Table 2. Access Matrix for room

Roles	P	B
Speaker	read,control	read,write
Listener	read	read,write

Speakers and listeners are allowed to read and write to the whiteboard, but only speakers can control the projector, i.e. move the slides forward or back.

A single user enters the room, with a credential attesting to his system role of `student`. The space starts a *session* with the mode set to `individual` and assigns the user a current_role of `student`. In this mode, the ALs show that this student is allowed to read and write the whiteboard W. This "virtual space" configuration is shown in Table 3.

Table 3. Individual Session

Access Matrix		
Role	P	B
Student	r	r,w
Faculty	r,c	r,w
Sysadm	r,c	r,w
Spaceadm	r,c	r,w

Current Role Translation		
user	sysrole	spacerole
u_1	student	student

Mode
Ind

When a lecture is scheduled in this space, other users enter the space. At least one of these users will be a *faculty* member, giving the lecture, while others may be students or other faculty members.

When this group assembles, the space automatically creates two new group roles—a *shared* role G_s with the intersection set of the permissions of all the members, and a *collaborative* role G_c that has all the permissions that any member of the group has. By default, the space switches into the *shared* mode. The current_role is set to G_s and the mode to `shared`. The state of the access control system is shown in Table 4.

Table 4. Shared Session

Access Matrix		
Role	P	B
Student	r	r,w
Faculty	r,c	r,w
Sysadm	r,c	r,w
Spaceadm	r,c	r,w
$\mathbf{G_s}$	**r**	**r,w**
$\mathbf{G_c}$	**r,c**	**r,w**

Current Role Translation		
user	sysrole	spacerole
u_1	student	shared
⋮	⋮	⋮
u_n	faculty	shared

Mode
Shared

Notice that in this *shared* mode, no one can control the projector (so it is not usable in this mode). The shared mode would be useful for a meeting, with shared access to the whiteboard.

To start the `lecture` application, the space must go into *supervised* mode. The faculty member giving the lecture has to re-authenticate, after which he is given the supervisor role, i.e., his current role is set to `Supervisor`, which has more permissions than the shared role. Once the `lecture` application is started, a new session is created in the space. Each user in the space is assigned a `current_role`, of either `speaker` or `listener`, and this mapping is stored in the current_role table. The state of the access control system is described in Table 5.

Table 5. Supervised Lecture Session

Access Matrix		
Role	P	W
Speaker	r,c	r,w
Listener	r	r,w
$\mathbf{G_{shared}}$	**r**	**r,w**
$\mathbf{G_{collab}}$	**r,c**	**r,w**
Supervisor	**r,c**	**r,w**

Current Role Translation		
user	sysrole	spacerole
u_1	student	listener
⋮	⋮	⋮
u_n	faculty	speaker

Mode
Super

When the lecture is over, the application session is over. Most participants leave the room. Suppose a few students decide to stay behind and have a meeting about a project for the class. The `current_role` table mappings are updated, as is the current space mode, and the space can turn into a meeting "virtual space". If there are only students present, the shared mode will have exactly the permissions for students.

To summarize, we have demonstrated the working of our access control system for an Active Space. We show how the same physical space with its set of devices and services can be reconfigured for different applications, such as a lecture or a meeting, and how access control is implemented for different types of shared-use modes of the room.

4. Access Control Model and Validation

An access control request has three parameters: a subject making the access request, a system object, and the specific object right (or method) being requested. Access rights to objects are traditionally stored in access lists (ALs) or capability lists (CLs). These two lists are conceptually merged into an access matrix [14], which is a table whose rows are subjects and columns are objects. Each entry in an access matrix contains a subject's set of rights on an object.

In practical access control systems, different sets of users may be allowed (or authorized) to create and delete subjects, objects, and rights in the access matrix. An access control policy is a set of rules that specify how the access matrix can be modified. Given an initial state of the access matrix containing only authorized access rights, the policy rules may be viewed as transitions that change the state of the access matrix. A policy specification is secure if the set of all reachable states are authorized, starting at the given initial state [6]. Alternatively, a secure access control system obeys the following safety property: the access matrix at any point of time contains only authorized access rights.

The objects in our model are the services 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 permissions for this service. The protection domains in our model are defined by the mode of the space.

We use RBAC as the primary access control mechanism in our system. The key concept in RBAC is a role, which is a placeholder for a set of users. Each role is associated with a set of permissions, which are its rights on objects. RBAC maintains two mappings: the User Role Assignment (URA) and the Role Permission Assignment (RPA), which can be updated independently.

4.1. Policy Types

We argue that any access control system for Active Spaces needs to support two types of access control policies: *mandatory* and *discretionary*. In a DAC (Discretionary Access Control) system, users are allowed to own objects and delegate access rights to these objects to other users. In a Mandatory Access Control (MAC) system, only the system administrator can modify the access matrix. In an Active Space, users are allowed to create DAC policies for devices that they own. However, the space administrator is responsible for defining access control policies for services that are part of the space and shared by the users. As a result, we provide support for both types of policies in our system.

In the access matrix for our system, subjects represent user roles, and objects are services in the space. The rights correspond to the list of methods exported by service interfaces. The policy specification defines the rules to change these entries in the access matrix.

In order to change an entry in the access matrix, users are required to produce a proof attesting that they are authorized to do so. We rely on three types of credentials that attest to the *identity* of entities (primarily users), the *ownership* of one entity by another, and the existence of a permission on a service. These credentials cannot be delegated. An example identity credential $\mathsf{typeof}(\mathsf{Joe}, \mathsf{administrator})$ asserts that Joe's role is administrator. The credential $\mathsf{exports}(\mathsf{service}, \mathsf{method})$ or $\mathsf{owns}(\mathsf{user}, \mathsf{service})$ attests that the method is exported by the service or the service is owned by the user, respectively.

4.2. Domains

Conventional operating systems often represent each user or each process as a protection domain. In our model, the protection domain of the Active Space is defined by the mode of the space, which in turn depends on the set of users in the space, as explained in Section 2.3. In each protection domain, a subset of all possible access rights are valid, and hence only that subset of operations can be performed. We have describe the process of mode selection and switching earlier; we now proceed to present a semi-formal specification of our access control model, followed by an informal proof of validation.

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.

For our policy specification, we use a guarded command notation, similar to the guarded command language[7, 18]. A guarded command is represented as a $(\mathsf{guard} \longrightarrow \mathsf{command})$ 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.

The specification of our access control policies is given below:

State Variables:

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 typeof(u : USERS; r : SYSTEMROLES) owns(u : USERS; o : OBJECT) exports(s : SERVICES; m : 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	
CRT	: **set of** $\{\langle u : \text{USERS}; r_{sys} : \text{SYSROLES}; r_{space} : \text{SPACEROLES}\rangle\}$ (Current role assignment for users in space AS)	

SysAdm $\in R_{sys}$, SpaceAdm $\in R_{sys}$

Functions

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

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

$\textbf{allow}(u, s, m) \wedge \text{typeof}(u, r) \in \mathcal{C}$
$\quad \wedge (s \in S) \wedge \text{exports}(s, m)$
$\quad \wedge (\text{currentrole}(r, mode), m) \in AL_s \longrightarrow$ **true**

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*.

$AddUser(u_{adm}, u) \wedge \text{typeof}(u_{adm}, \text{SysAdm}) \in \mathcal{C}$
$\quad \longrightarrow U := U \cup \{u\}$

Specifications for RemoveUser, AddSysRole, RemoveSysRole, AddSysService, 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.

$AddPermToSysRole(u_{adm}, r, s, m) \wedge$
$\quad \text{typeof}(u_{adm}, \text{SysAdm}) \wedge (r \in R_{sys}) \wedge (s \in S) \wedge$
$\quad (m \in M) \wedge \text{exports}(s, m) \longrightarrow AL_s := AL_s \cup \{\langle r, m\rangle\}$

RemovePermFromSysRole is similar and not shown.

The system policy specification should specify 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.

$\forall r \in R_{sys} : R_{space} := R_{space} \cup \{r\}$

$AddSpaceRole(u_a, r) \wedge \text{typeof}(u_a, \text{SpaceAdm})$
$\quad \longrightarrow R_{space} := R_{space} \cup \{r\}$

RemoveSpaceRole(u_a, r_{space}) is similar and not shown.

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

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

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

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

$CreateCollabRole(CRT) \longrightarrow$
$\quad \forall AL_s : s \in AS$ (all ALs in space),
$\qquad$ if $\exists u \in CU, (\langle u, role\rangle \in \mathcal{C}) \wedge \{\langle role, m\rangle\} \in AL_s$
$\qquad$ 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.

$EnterSpace(u) \wedge CU \neq \phi$
$\quad \wedge \text{typeof}(u, r) \in \mathcal{C}$
$\quad \longrightarrow \quad CRT := CRT \cup \langle u, r, r\rangle\}$
$\qquad\qquad CreateSharedRole(CRT)$
$\qquad\qquad CreateCollabRole(CRT)$
$\qquad\qquad SwitchToSharedRole(CRT)$
$\qquad\qquad CU := CU \cup u$

$SwitchToSharedRole(CRT)$
$\quad \forall u \in CU$
$\quad \wedge \text{typeof}(u, role) \in \mathcal{C}$
$\quad CRT := CRT \cup \{\langle u; role; r_{shared}\rangle\}$
$\quad \text{Mode} = Shared$

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

$EnterSpace(u) \wedge u \in CU \wedge \text{typeof}(u, r) \in \mathcal{C}$
$\quad \longrightarrow \quad CU := CU - u$
$\qquad\qquad CRT := CRT - \langle u, r, r\rangle\}$
$\qquad\qquad CreateSharedRole(CRT)$
$\qquad\qquad CreateCollabRole(CRT)$
$\qquad\qquad SwitchToSharedRole(CRT)$

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

Supervisor
$GetSuper(r, s, m) \wedge (Mode = \mathsf{Super}) \wedge \text{typeof}(r, \mathsf{super}) \wedge$
$(\{\langle r, s, m\rangle\} \in AL_s) \longrightarrow$ **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.
$map(u_{sp\_adm}, r_a, r_{space}) \wedge \text{typeof}(u_{sp_adm}, \mathsf{SpaceAdm})$
$\longrightarrow R_{space} := R_{space} \cup \{r_a\}$

Discretionary Access Policies For every "owned" device D that is to be allowed in the system, the system administrator must first create the 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.

```
$AddOwnedDevice(u_{adm}, owner_d, d) \wedge (u_{adm} \in \mathsf{SysAdm})$
  $\wedge\ (owner_d \in U) \longrightarrow$
      $S := S \cup \{d\}$
      $OD := OD \cup \{d\}$
      $AddSysRole(u_{adm}, \text{DeviceOwner}_d)$
      $AddUserToSysRole(u_{adm}, owner_d, \text{DeviceOwner}_d)$
      $\mathcal{C} := \mathcal{C} \cup \text{owns}(owner_d, d)$
$AddDeviceRole(u_d, d, r_d) \wedge \text{owns}(u_d, d) \longrightarrow$
  $R_{dev} := R_{dev} \cup r_d$
$AddPermToDevRole(u_d, r_d, d, m) \wedge \text{owns}(u_d, d)$
  $(d \in OD) \wedge exports(d, m) \wedge (r_d \in R_d)$
      $\longrightarrow AL_d := AL_d \cup \{\langle r_d, m\rangle\}$
$AddUserToDevRole(u_d, u, r_d) \wedge \text{owns}(u_d, d)$
  $\wedge (u \in U) \longrightarrow URA_d := URA_d \cup u_d$
```

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 $typeof, owns$, or $exports$ 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 $exports$. 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 $owns(owner_d, d)$ 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.

5. Related Work

Many research projects are working to provide infrastructure support for using interactive workspaces [4, 12, 16]. While a lot of research has gone into adapting traditional applications to an environment of heterogeneous devices [24], the security issues have not received as much attention. We believe that Active Space environments cannot be deployed for non-trivial use until a security architecture is in place.

Access control mechanisms are required whenever objects must be shared in a controlled manner. The access matrix model [14] is the most popular one, used in most operating systems. Systems such as UNIX use access control lists and discretionary access controls—owners of objects can assign others with rights to them. The problem with such environments is that information flow is hard to analyze. Military systems, therefore, use MAC systems, where the "system administrator" controls and sets policy for access to all system objects. DTE [1] provides a domain-type enforcement environment for UNIX, and TrustedSolaris [22] and SELinux [20] provides support for MAC policies on Solaris and Linux, respectively.

Shen [19] proposed an access control architecture for collaborative environments based on a generalized editing model of collaboration. The most popular techniques for access control in recent years have been variants of RBAC [8, 17]. These techniques have been adapted for use in ubiquitous computing environments [9, 23, 5], and the concept of roles is extended to deal with context information. While networked applications have traditionally attempted to hide physical location, by providing uniform interfaces for local and remote users to access services [2], in intelligent environments such as smart rooms [15], spatial location is often important to the organization of communication [11].

6. Conclusion

We present an access control architecture for Active Spaces. The novel features of our model include the integration of physical and virtual access control mechanisms, the ability to dynamically reconfigure access control policies to create different policies for different virtual spaces, and a semi-formal model with validation of the safety property that all access control decisions in our system are authorized.

We introduce the notion of collaborative access control modes and develop appropriate access control policies that augment traditional modes of collaboration and enable new models of interactive behavior for active applications. An Active Space may be in an individual, group supervised, or collaborative mode, depending on the users, devices, and other context in the space. These modes also define different protection domains. Changing between modes preserves the tranquility properties of authorizations, except for the special case of collaboration that explicitly provides mechanisms for rights amplification within a mutually trusting group of users.

Our system uses the RBAC model for policy configuration, and implements both discretionary and mandatory access controls, providing flexibility and expressiveness. We define three types of roles viz., system, space, and application roles. Each role can be managed by a different administrator, thus allowing for decentralized administration, and most of the policy configuration tasks are automated. We describe our prototype implementation and show how this role classification provides a clean separation of duties and simplifies the configuration, implementation, and enforcement of access control policies for dynamic environments exemplified by Active Spaces.

References

[1] L. Badger, D. F. Sterne, et al. A domain and type enforcement UNIX prototype. In *Proceedings of the 5th Usenix UNIX Security Symposium*, Salt Lake City, Utah, June 1995.

[2] M. Beigl and H.-W. Gellersen. Ambient telepresence. In *Proceedings of the Workshop on Changing Places*, pages 63–69, London, UK, 1999.

[3] K. J. Biba. Integrity considerations for secure computer systems. Technical Report MTR-3153, Mitre Corporation, 1975.

[4] R. Cerqueira, C. K. Hess, et al. 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, Sept. 2001.

[5] M. J. Covington, M. J. Moyer, et al. 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.

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

[7] E. Dijkstra. Guarded commands, nondeterminacy, and formal derivation of programs. Communications of the ACM, 18(8):453–457, 1975., 1975.

[8] D. F. Ferraiolo and D. R. Kuhn. Role-based access controls. In *Proceedings of the 15th NIST-NSA National Computer Security Conference*, Baltimore, MD, oct 1992.

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

[10] T. O. M. Group. Common object request broker architecture (CORBA/IIOP).

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

[12] B. Johanson, A. Fox, et al. The Interactive Workspaces project: Experiences with ubiquitous computing environments. *IEEE Pervasive Computing magazine*, 1(2), apr–jun 2002.

[13] T. Kindberg and A. Fox. System software for ubiquitous computing. *IEEE Pervasive Computing*, 1(1):70–81, jan–mar 2002.

[14] B. W. Lampson. Protection. In *Proceedings of the 5th Princeton Symposium on Inforamtion Sciences and Systems*, pages 437–443, Princeton, NJ, Mar. 1971. Princeton University.

[15] A. P. Pentland. Smart rooms. *Scientific American*, Apr. 1996.

[16] M. Román, C. K. Hess, et al. GaiaOS: A middleware infrastructure to enable active spaces. *IEEE Pervasive Computing (to appear)*, 2002.

[17] R. S. Sandhu, E. J. Coyne, et al. Role-based access control models. *IEEE Computer*, 20(2):38–47, 1996.

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

[19] H. Shen and P. Dewan. Access control for collaborative environments. In J. Turner and R. Kraut, editors, *Proceedings AC Conference on Computer-Supported Collaborative Work (CSCW)*, pages 51–58. ACM Press, 31 –4 1992.

[20] S. Smalley and T. Fraser. A security policy confi guration for security-enhanced linux. Technical report, NAI Labs, 2001.

[21] J. P. Sousa and D. Garlan. Aura: an architectural framework for user mobility in ubiquitous computing environments. In *Proceedings of the 3rd Working IEEE/IFIP Conference on Software Architecture*, Montreal, aug 2002.

[22] Sun Microsystems. Trusted Solaris.

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

[24] P. Werle, F. Kilander, et al. A ubiquitous service environment with active documents for teamwork support. In *Proceedings of Ubicomp 2001*, Atlanta, GA, September 30 – October 2 2001.