CONTEXT-AWARE ACCESS CONTROL IN PERVASIVE COMPUTING ENVIRONMENTS

By

TAMMY NGUYEN

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

MASTER OF SCIENCE IN COMPUTER SCIENCE

WASHINGTON STATE UNIVERSITY
School of Electrical Engineering and Computer Science

AUGUST 2005

To the Faculty of Washington State University:

The members of the Committee appointed to examine the thesis of TAMMY NGUYEN find it satisfactory and recommend that it be accepted.

Chair

ACKNOWLEDGEMENT

I would like to thank the school of Electrical Engineering and Computer Science for providing teaching assistant support. I would also like to thank Dr. Murali Medidi for guidance on my thesis and in my classes. I would also like to acknowledge Jack Hagemeister for all his support during my graduate studies. I would also like to thank Dr. K. C. Wang for his influence and support throughout both my undergraduate and graduate studies.

CONTEXT-AWARE ACCESS CONTROL IN PERVASIVE COMPUTING ENVIRONMENTS

Abstract

by Tammy Nguyen, M.S.
Washington State University
August 2005

Chair: Muralidhar Medidi

Pervasive computing requires participating services and applications to be non-intrusive and seamless when interacting with users requiring applications to become aware and adapt to the context of an environment. Although awareness is available through contextual information, applications will only gather information that best fits their function. Hence, entities may become unaware of intrusions or conflicts that may or will be occurring, which could affect one's performance, confidentiality, and integrity. We present a context-aware access control model that provides a representation of user and service accesses based on gathered context. The model incorporates users, services, permissions, and activities become the main factors for determining whether access is granted or denied. Based on context constraints, access control policies evolve to fit the security requirements of the active space.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

Pervasive Computing integrates the physical environment space with the user space. It allows users to interact with the environment in a way that allows users to reduce their focus on computing technology and concentrate more on their current tasks. Designing a pervasive system requires integration of all areas of computer science and engineering from hardware designs to theoretical studies. The area of Pervasive Computing which this research addresses is access control. Though the system would eventually be designed and implemented, security and trust issues could prevent it from being used. Users will interact with the smart environment with interactive applications on peripheral devices that communicate with the system providing services to their current task. The environment contains applications or brokers waiting for requests of service to carry out tasks of which it was designed. Access control to various objects, files, or devices becomes necessary to the success of Marc Weiser's vision [69]. This vision focuses on the seamless interaction between users and the environment filled with embedded computing systems. These objects could be any household or office appliance, electronic files, or peripheral devices. In general, the objects requiring protection are physical devices or virtual files. Pervasive Computing brings other issues that raise concern because of its high mobility and service capabilities. One could request a service that wishes access to a resource without regard for activities occurring in the environment. Such a request could violate the user intentions of the current activity. Actions by users in a space continuously affect the security properties of a smart environment where they cannot always see or hear all actions or events occurring [10]. Thus, Pervasive Computing brings the issue of preserving user intents in a physical environment where users are consistently interacting with the space while still applying the appropriate security policies and preventing unauthorized accesses, [10].

For example, consider an instructor in a classroom lecturing to her students. The instructor would not want the lights to turn off unexpectedly when the lecture needs a well-lit room. In a typical environment, at a minimum, users are normally allowed the permission to turn on and off the lights of rooms they are allowed access. However, in this scenario violation of the instructors objectives should be prohibited. In a pervasive environment the lighting would be controlled by a lighting service. If suitable controls are not in place that evaluates the context of the environment (did the request originate in the room?) then a student or a user external to the current activity could access the light control remotely, and turn off the lights.

Further expansion on the lighting example demonstrates other cases that must be accounted for. The typical building is designed to conserve energy by automatically shutting off most lights at the end of the day and to increase security by locking the entrances. This saves the buildings occupants the expense of lighting the building when nobody is there. However, when someone stays late, or an unplanned function occurs that causes people to need access outside of normal hours, they are inconvenienced and will need to find the manual override for the lights and most likely prop open the entrance for others to enter the building. This does not allow for seamless interaction but creates an annoyance because it does not consider the user's intent, [53], especially when no harm is done by working a little extra. Therefore, the access control mechanism in pervasive environment must preserve user intent by preventing services from performing actions that my contradict user intent and current context.

1.1 Challenges

Although there has been much work done for context-awareness in user applications and architectures, few projects have addressed the issue of conflicting inference of user intent in the multiuse of context-aware applications. Most projects approach Pervasive Computing from the view of providing services for users. These services may adapt to the context of the environment without the precautions needed to integrate various applications. It may attempt to request the same service or conflicting services that may expose the environment to unexpected behaviors. We are developing security techniques that are flexible and nonintrusive for Pervasive Computing environments.

Most current applications require user interaction through explicit requests or inputs. As users move away from these desktop applications, applications may need to infer user intent from collected context and the few explicit demands the user has previously made. The pervasive environment is information-rich, filled with context data, while users move about with their mobile devices and interacting with the environment system. Applications in these environments will process implicit information about users to adapt and provide services as needed. Through the processed information services may perform actions that affect the state of the environment, and it may be unknown whether these acts fit the context of the environment. In a pervasive environment the user applications and the services provided by the environment are tightly coupled, but are not necessarily coordinated. This makes it a challenge to provide services for the user that will respect the state of the environment

Pervasive Computing environments needs an access control system to operate efficiently and effectively to provide for its dynamic and transparent nature. In such environments users are continuously interacting with the system; retrieving and sending information and requests for various services. All processing of information and requests for services must be done rapidly and transparently, and all information must also be readily available for users. Furthermore, it is necessary for the system to determine the origin of the request of information or service. Also, collected context information can be used to provide a more fine-grained policy. Hence, our goal for this thesis is to provide an access control model that is able to accommodate for these information-rich computing environments.

Access control restricts entities only to their authorized information or resources, providing confidentiality and integrity. In PC environments it must support a descriptive and flexible policy that incorporates gathered context information and physical security awareness [1]. Access control requirements should be dynamic and adjust to the context of the environment. Many access control mechanisms today are unable to include basic context requirements in their policy and gather enough contexts for analysis. To clarify: context is collected information that characterizes activities encapsulating an entity, where an entity could be any person, place, or object [18]. Collection of context must be secure and protected to prevent bogus information from being sent to the system. The system must also be responsive to services and provide enough detail so services can determine a way to handle changes in access and remain unobtrusive

As PC environments are highly adaptive and unpredictable it would not be possible for a user to identify all the potential security requirements and risks for a given situation without any assistance [68]. Hence, a PC environment is dependent on access control policies and mechanisms to protect users from misbehavior or inconsistencies while the context and application behavior changes.

1.2 Contributions

This thesis shows it is possible to provide a context-aware access control service for Pervasive Computing environments to both preserve user intentions and secure context-aware services. The work allows for flexibility of well-defined policy definition based on subjects, roles, permissions, activities, and context constraints. The implementation shows that it is possible to provide a system to provide such services.

1.2.1 Contributions

This thesis provides evidence to the following areas:

Activity-based Access Control Model An access control model provides an abstraction to view implementation of security policies. We present a flexible access control model that allows for a generalized policy implementation of security policies allowing for context constraints. Particularly, we present an activity-based access control model that allows for policies to be specified in terms of activities occurring in an environment, where context constraints specifies the contextual conditions where the activities may occur. We describe the activity-based access control model and the various components in the model.

Activity Role Constraints Activity role constraints provide a technique to restrict certain subjects from joining an activity. It prevents unauthorized participation in various activities. Also, it contains a requirement of participating roles before an activity activation.

Mutually Exclusive Activity Constraints Often, two or more activities may be disjoint meaning that if activity α is activity, then activity β cannot be activated. Providing mutual exclusion between activities contributes in the control of activated activities.

Activity Context Constraints The context of an environment must be realized to determine whether activities are activated correctly. Activity context constraint provides a mean to constrain activities to the context of an environment.

Activity Role Context Constraints In various cases, some context in an environment may allow an activity to be activated and some may disable the activity. Context constraints are associated with some set of roles. To deactivate an activity based on noncritical constraints may cause a nuisance. We then constrain both activities and activity roles based on some context constraint, distinguishing between critical and noncritical constraints.

Access Control Implementation We discuss an implementation and show how it could be used to preserve user intentions and secure context-aware services.

1.2.2 Thesis Outline

The remainder of this thesis is organized as follows:

Chapter 2 presents an in-depth discussion of current access control models for Pervasive Computing. It also discusses the reasons these access control models are not sufficient for Pervasive Computing systems.

Chapter 3 introduces the Activity-based Access Control model that supports context constraint definitions for access control policies. It provides a redefinition of the role-based access control model to incorporate the different entities in pervasive environments. The chapter presents the four types of constraints: Activity-based Access Control Model, Mutually Exclusive Activity Constraints, Activity Context Constraints, and Activity Role Context Constraints.

Chapter 4 builds upon our description of chapters 3 through implementation of the various security services and an illustration of how they can be used to preserve user intentions while securing context-aware applications.

Chapter 5 provides a summary, conclusion, and suggested future work.

Chapter 2

Background & Related Works

Formal models are intended for security designers to persuade others of the system security [34]. A security designer must be able to make a convincing argument that their system is secure. A convincing argument usually starts with constructing a model and consists of two parts: (1) demonstration that a system conforming to the model is secure, and (2) demonstration that the system design enforces the model.

In the past, the main purpose of computers was to store and modify information to edit easily, distribute, and read messages and documents. Shared computers required access control as a means to protect documents from being accessed by unauthorized users. Landwehr [34] stated that "most of the difficulties arise precisely because a computer shared by several users cannot be viewed as a passive object in the same sense that a safe or a pencil is passive." The example given by Landwehr describes a program displaying a part of a document. The user viewing the document may not be the author, but the program author may have made the document accessible to others without the knowledge of the user. Such a behavior would then cause a violation. For example, if an unauthorized user reads the document, the user's security would be violated.

Additionally, most multiuser systems require recording of a document's security level. This requires the cooperation of multiple programs which may be written by many different developers in many different languages using different compilers. It becomes more complex to set the security level and classification of a virtual object than a simple physical notebook.

Landwehr [34] classifies three types of threats that automated systems must be protected: (1) the unauthorized disclosure of information (2) the unauthorized modification of information, and (3) the unauthorized withholding of information, otherwise known as a denial of service (DoS). Relative to access control, the following problems may occur: inference, browsing, integrity, copying, denial of service, confinement, Trojan horses, and trapdoors. The problem of inference forms when queries for data or objects provide much more information than needed for a user to determine that certain objects are restricted. This may result in either the user thinking a resource is secured when it is not, or thinking that a resource has been made available to others when it has not. Additionally, restriction may be needed based on the amount of information a user may browse. Usually, it is unnecessary for a user to know the existence of every possible resource. By knowing an object's existence, the user may secretly access the objects that she knows she would have the authorization, even though she would not have known of its existence without the computer's assistance. This is known as the browsing problem. The integrity problem occurs when users make unauthorized modifications to an object. Such modifications are difficult to detect on a computer. The copying problem occurs when a file copied to another location is unauthorized and undetected. Denial of service is to prevent access when the user should have access, but because of some circumstance such as a malfunction, a user may be prevented access.

Typically a program, when executed, acts on behalf of the user. A user requests for the execution of a program, the process associated with the user executes the program, and the program accesses information on behalf of the process. The accessibility rights of a program are restricted by the permissions allowed by the user. Unfortunately, information accessed by programs may leak the data to other processes or files which are accessible to other users [33]. This problem is known as the confinement problem discovered by Lampson.

A Trojan horse on a user's machine or the existence of a trapdoor would compromise a system. A Trojan horse is a masquerading attack where a program pretends to be a service, but surreptitiously sends data to others, while a trapdoor is hidden code that responds to certain input to bypassing typical security measures.

As computers become more advanced, these problems that existed in the past will typically always remain, and due to additional possible capabilities in technology, more problems will arise. In this chapter, we discuss models that were developed to help overcome some of these problems and models directed towards PC environments.

2.1 Access Control Background

It is not easy to capture real world complexities into a formal model [34]. Since the real world is perceived from many different perspectives, and security systems may have different purposes, security models chosen for a system are usually based on one's understanding of the model and how the model fits the purpose of the system they are designing. Hence, there are many different access control models that were developed each from a different perspective and level of detail. The model provides an abstraction of the real world, but enforces controls that are much stricter than in the current environment. In the following subsections, we discuss basic access control terms as well as the following models: the Access Control Matrix, Discretionary Access Control, Mandatory Access Control, and Role-based Access Control.

2.1.1 Basic Terms and Definitions [1]

Definition 2.1.1.1. A policy defines rules and guidelines describing accesses that are allowed to be authorized.

Definition 2.1.1.2. An access control mechanism implements the policies through software and hardware.

Definition 2.1.1.3. A formal access control model defines the access control specification and enforcement. The model is required to be complete and consistent. It must encompass all security requirements and contains no contradictions such that there is no case where an access control mechanism could both disallow and grant access simultaneously. Proving a model is "secure" and a mechanism accurately implements the model, a designer may argue the system is secure.

Definition 2.1.1.4. Access is a specific type of interaction among a subject and an object that results in the flow of information from one to the other.

[1] Terms and Definitions provided by DoD Trusted Computer System Evaluation Criteria [16]

Definition 2.1.1.5. Access control is the process of limiting access to the resources of a system only to authorized programs, processes, or other systems (in a network).

Definition 2.1.1.6. An access control mechanism is hardware or software features, operating procedures, management procedures, and various combinations of these designed to detect and prevent unauthorized access and to permit authorized access in an automated system.

Definition 2.1.1.7. Authorization is the granting of access rights to a user, program, or process.

Definition 2.1.1.8. Availability of data is the state when data are in the place needed by the user, at the time the user needs them, and in the form needed by the user.

Definition 2.1.1.9. Confidentiality is the concept of holding sensitive data in confidence, limited to an appropriate set of individuals or organizations.

Definition 2.1.1.10. Data integrity is the property that datum meet a priori expectation of quality.

Definition 2.1.1.11. An object is a passive entity that contains or receives information. Access to an object potentially infers access to the information it contains.

Definition 2.1.1.12. Permission is a description of the type of authorized interactions a subject can have with an object.

Definition 2.1.1.13. A security policy model is a formal presentation of the security policy enforced by the system. It must identify the set of rules and practices that regulate how a system manages, protects, and distributes sensitive information.

Definition 2.1.1.14. Security requirements are the types and levels of protection necessary for equipment, data, information, applications, and facilities to meet security policy.

Definition 2.1.1.15. A subject is an active entity, generally in the form of a person, process, or device, which causes information to flow among objects or changes the system state.

Definition 2.1.1.16. A user is a person or process accessing a computing system.

2.1.2 Access Control Matrix Model

The access control matrix is a basic framework to describe relationships between subjects and objects in a system. For example, this model is used in both operating systems and database systems. An access control matrix is a systematic way of describing shared data or resources and controlling the access to these shared objects. Lampson [32] defined a system with a set of objects O, a set of subjects S and an access function A which is called an access matrix . An object is an entity to be protected, a subject is an entity which has access to an object and which subjects can access which objects are determined by the access matrix. The rows of the access matrix are labeled by the subjects and the columns by the objects, thus element A[i, j] specifies the access subject i has to object j. An element consists of a set of strings that specifies what type of access a subject has (access attributes or access permissions). For example read and write are typical access attributes but others can be specified as well.

In example 2.1.2.1, Subject 1 has read access to file 1 and has owner access to both itself and Subject 2. It shows that a subject can also be an object if access to a subject must also have control.

Tab. 2.1: Access Control Matrix Example[1]

Example 2.1.2.1.

-	Subject 1	Subject 2	Subject 3	File 1	File 2	Process 1
Subject 1	owner,control	owner,control	call	owner,read,write		
Subject 2			call	read	write	wakeup
Subject 3			owner,control	read	write	

[1] Adaption of Lampson example [32]

2.1.3 Discretionary Access Control Model

In discretionary access control (DAC) it is left to the owners own discretion to determine the access to her files. This is the most common type of access control. For example, suppose a person is writing a technical report. She controls who is allowed or not allowed to read the report. Her managers are allowed to read it, but not her co-workers or family. If an individual user can set an access control mechanism to allow or deny access to an object, that mechanism is a discretionary access control (DAC).

2.1.4 Mandatory Access Control Model

Access control where the owner can not control who can access the owned objects is called a mandatory access control. When a system mechanism controls access to an object and an individual user cannot alter that access, the control is a mandatory access control (MAC). Information about both the subject and the object, which the subject wants to access, is used to determine whether the access is allowed or not.

2.1.5 Role-based Access Control

The role-based access control (RBAC) model, shown in Figure 2.1 is an access control model, which involves the concepts of users, roles, sessions, and permissions [21]. Subjects are typically represented by users according to the original Role-based Access Control model, but a process that acts on behalf of the user can also be a subject. The figure corresponds to the NIST model, which considers the user entity instead of subject. The communication instance between the user and the system is a session. Objects are resources that are accessible by a computer system such as files or devices. An operation is an active process invoked by the subject, and permissions are authorizations to perform operations on objects.

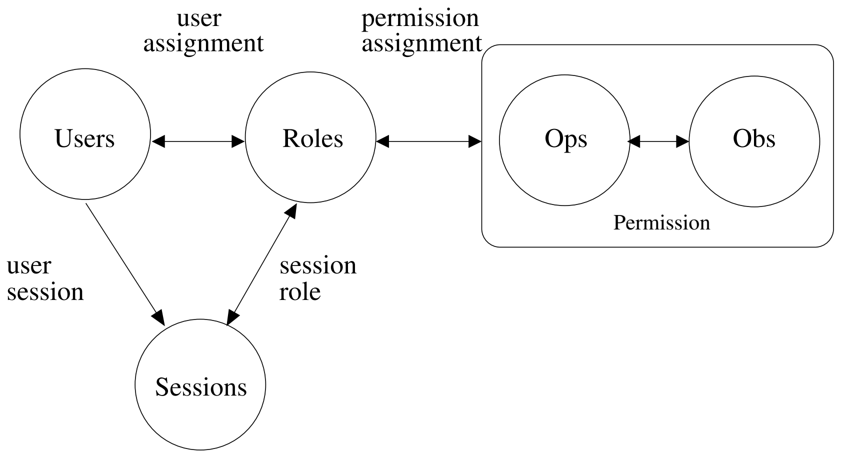

Fig. 2.1: Role-based Access Control Core Model (NIST model)

RBAC applies the concept of least privilege, which is an administrative action of avoiding the assignment of permissions to users which are unnecessary to accomplish a task. This prevents users from being given no more permissions than needed. It is this notion of least privileges that allows the distinction among various roles.

In RBAC, roles are assigned to users and permissions are assigned to roles. Roles are often confused with groups, but there is a clear distinction between the two concepts. The assignment of roles to users gives the user the collection of permissions assigned to the roles allowing for ease of maintainability. The subject performs the requests made by the user, which is identified with a unique identifier to determine if it is authorized for a role and can become an active role. The main idea of RBAC is to describe access policies according to some organization structure. For access control, it is not important to identify the individual who makes the access request, but to know her position in the structure of the organization. It groups privileges or authorizations that can be applied to a group of users. This concept is known as a role in RBAC. A *role* is a job function within the context of an organization with some associated semantics regarding the authority and responsibility conferred on the user assigned to the role. Intuitively users are assigned roles, and permissions are assigned to roles.

The RBAC model can be divided into several model components, where each provides some specific features: (1) Core RBAC (2) Hierarchical RBAC (3) Constrained RBAC. The following subsections describe the three components of the model and define the main characteristics of RBAC. Each model is described in terms of a set of basic element sets, relations, and mapping functions.

2.1.5.1 Core RBAC

The most basic RBAC model is *core* RBAC. It provides a foundation for the entire model and its components. The basic elements are summarized as follows:

- *Users* (USERS) are human beings. It is important to know that it may be extended to represent processes, or any active entity of the system.
- *Roles* (ROLES) are job functions within the context of an organization with some associated semantics regarding the authority and responsibility conferred on the user assigned to the role.
- *Objects* (OBS) are data or resources to be accessed.
- *Operations* (OPS) are processes that execute some function on behalf of the user.
- *Permissions* (PRMS) are defined as an approval of a particular mode of access to one or more objects in the system. Permissions establish a relation between operations and objects. That is, they define the operations to be performed over objects. Thus, the permissions are expressed as the following:

$$PRMS = 2^{OPS \times OBS}.$$

That is, $2^{OPS \times OBS}$ denotes the powerset of the set of $OPS \times OBS$. $OPS \times OBS$ denotes the cartesian product of operations and objects.

The core RBAC model defines relations between the following objects:

- *User Assignment* (UA) defines the relations among users and roles. Note that a user may be assigned to more than one role, and several users can be assigned to the same role. It is a many-to-many mapping, where:

$$UA \subseteq USERS \times ROLES$$

- *Permission Assignment* (PA) defines the relation between roles and permissions. Here, a role may be assigned to multiple permissions. Similar to UA, it defines a many-to-many mapping:

  $$PA \subseteq PRMS \times ROLES$$

In addition, RBAC defines a mapping between users and roles as a set of sessions (SESSIONS). The idea is that when a user logs into the system, the user may be mapped to one or several roles through a session. This allows a user to activate only the minimum subset of roles they need to perform a specific task, and hence by using sessions, RBAC supports the least privilege principle. The mapping is established by two functions: *session_users* and *session_roles* [21].

2.1.5.2 Separation of Duty

The concept of separation of duty applies constraints on roles in a session so it reduces the number of potential permissions available to a user. This provides extensibility to the principle of least privilege. The role hierarchy (roles containing other roles) extends the role layer to multiple layers, reducing the number of relationships that are managed. It helps manage role complexity through structure to exploit commonality not only between users but among roles. Role hierarchies allow a policy implementer to write generic access rules just once, rather than for every role to which the rule applies.

2.1.6 General Role Hierarchy

The *general role hierarchy* supports multiple inheritance. That is, a role can inherit permissions from more than one role. Given the permission inheritance, another inheritance relation regarding the users can be defined. The role hierarchy states that if a subject is authorized to access a role and that role contains another role, then the subject is also allowed to access the contained role. In some cases, multiple inheritances can introduce conflicts and problems similar to those of multiple inheritances in object oriented languages. Limited role hierarchy restricts the general role hierarchy by not allowing multiple inheritances. For instance, a role such as a project coordinator will not be able to inherit permissions from two different roles if limited role hierarchy is enforced.

2.1.7 Constrained RBAC

The constrained RBAC model adds *separation of duty* relations to RBAC. The idea is to prevent a user from gaining authorizations of permissions associated with conflicting roles. It supports static and dynamic separation of duty.

2.1.7.1 Static Separation of Duty

The *static separation of duty* (SSD) enforces constraints in the role assignment. Here, a user is authorized as a member of a role only if that role is not mutually exclusive with any of the other roles for which the user already possesses membership. An example of SSD is when the definition of two roles (or sets of roles) is mutually disjoint such that the same user cannot be assigned to both roles. Although there can be different types of SSD, the models just supports the definition of a set of roles and the ability to form user assignment relations. If a user is assigned to one role of the set, the user cannot be a member of another role in the set. Specifically, SSD takes two arguments:

- set of two or more roles, and
- cardinality greater than one. A user can be assigned to a number of roles from the set not exceeding the cardinality.

SSD can be combined with role hierarchy. The inheritance defined by the role hierarchy may bypass SSD policies. To avoid the bypass, role hierarchies include the inheritance of SSD constraints.

2.1.7.2 Dynamic Separation of Duty

While static separation of duty place constraints on the role assignment, *dynamic separation of duty* (DSD) places constraints on the activation of roles. That is, it limits the roles that a user can activate across a session. A user can become active in a new role only if the proposed role is not mutually exclusive with any of the roles in which the user is currently active. Additionally, a user can execute an operation only if the operation is authorized for the role in which the user is currently active, and a user can execute an operation only if the subject is acting within an active role.

2.2 Access Control Models for Pervasive Computing

2.2.1 Context-based Team-based Access Control Model

The work done by Georgiadis *et al.* [24], shown in Figure 2.2 is an integration of both the team-based access control model (TMAC) and RBAC. C-TMAC is still in its infancy, but it utilizes contextual information to filter out access requests. The TMAC model [60] was developed to provide an access control model for collaborative activities accomplished by teams of users. Thus, the TMAC introduces the notion of "teams" as a collection of users in specific roles. Teams collaborate with one another with the objective of accomplishing a specific task or goal. Users are assigned to teams, where each team will encompass the set of roles. Each user's access for team resources depends on his current role and current team. C-TMAC extends this idea to allow context information to tailor to fine-grained, flexible and context-based access control policies. For example, a policy may consider a nurses' shift when accessing patient records.

C-TMAC is formalizes as follows:

- U, R, P, S, T, C stands for sets of users, roles, permissions, sessions, teams and context
- $PRS \subseteq P \times R$ is a many-to-many permission to role assignment relation
- $URS \subseteq U \times R$ is a many-to-many user to role assignment relationship
- $CTS \subseteq C \times T$ is a many-to-many context to team assignment relation
- $UTS \subseteq U \times T$ is a many-to-many user to team assignment relation
- session-user: $S \rightarrow U$ is a function mapping each session s to the single user $user(s_i)$ that is constant for the session's lifetime

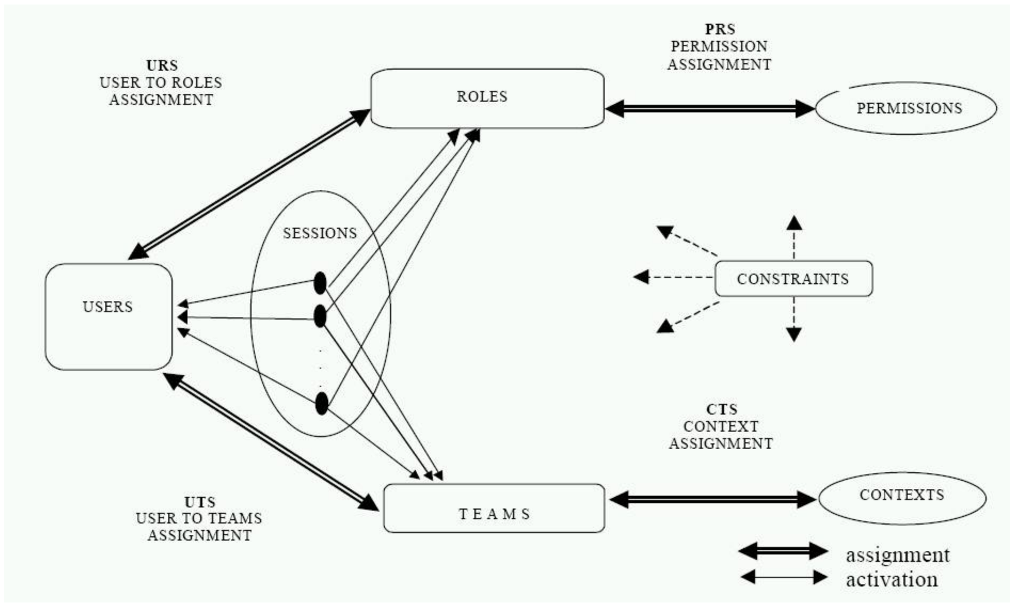

Fig. 2.2: C-TMAC Model

- session-roles: $S \rightarrow 2^R$ is a function mapping each session s_i to a set of roles $roles(s_i) \subseteq \{r|user(s_i), r) \in URS\}$, which can change with time, and session s_i has the permission $\cup r \in roles(s_i)p|(p,r) \in PRS$ referred to as Session-Roles Permissions
- session-teams: $S \rightarrow 2^R$ is a function mapping each session s_i to a set of teams $teams(s_i) \subseteq \{t|users(s_i), t) \in UTS\}$, which can change with time, and session s_i has the contexts $\cup t \in teams(s_i)\{c|(c,t) \in CTS\}$ referred to as Team-Context.
- team-users: $T \rightarrow 2^U$ is a function mapping each team t_i to a set of users $users(t_i \subseteq \{u|(u,t_i) \in UTS \wedge \exists s_j : user(s_j) = u\}$, which can change with time
- team-roles: $T \rightarrow 2^R$ is a function mapping each team t_i to a set of roles $roles(t_i) \subseteq \{r|(users(t_i), r) \in URS\}$, which can change with time, and team t_i has permissions $\bigoplus r \in roles(t_i)\{p|(p,r) \in PRS\}$ referred to as Team-Roles Permissions, and where $\bigoplus$ means

"combinations".

This model is an extension of RBAC, an inheritance from TMAC resulting in application of RBAC features. There are a few new concepts, which we will explain. Context (C) is information regarding the data object. Team (T) represents a group of users with specific roles with some objective or goal. Context assignment (CTS) is context information assigned to teams, and team assignment (UTS) is member assignment of teams.

Activations in C-TMAC require role selection and team selection. Role selection was inherited from RBAC where a user is required to activate a set of roles before she is allowed to utilize those permissions. In C-TMAC, after role selection the user would select a subset of teams to participate. The team context permissions filters out permissions based on specified conditions, and hence, the final user permission set is filtered based on the team's context.

2.2.2 Access Control for Active Spaces

The access control model developed by Sampemane *et al.* [50] allows for the dynamic protection of resources to preserve safety properties on a need to protect basis. The protection domain of the Active Space is defined by the mode of the space. In each protection domain, only a subset of all possible access rights is valid. They used the term Active Space to represent an interactive environment consisting of the physical spaces integrated with embedded computing and communication hardware and software. The term Active Space is interchangeable with other terms, such as "smart space," "Pervasive Computing environment", "intelligent environment", etc.

Users in their Active Space may need to explicitly state the mode of the space in some instances of the space. These modes are:

1. Individual user mode. In this mode, a user is allowed access to all the space resources that the access control policies allow in his space role.
2. Group modes. Here, each member of the group is granted the same permissions. There are two group modes where permissions are assigned to users, they are:

    - Shared mode. A group of users shares the space without any special trust relationship. The set of permissions allowed are only the intersection set of their individual permissions. Hence, a user may have his set of allowed permissions decreased when in this space mode.
    - Collaborative. In the trust mode, a group of users may collaborate with one another on a particular application. The users trust the people they are working with, and delegate their permissions to the group. The set of valid permissions is the union of the permission set of all individual members. All members would have the same set of permissions, but an individual may have gained more permissions than allowed.

    The space reacts to changes in the count of users in the space. If a single user is in individual use mode and a second user enters the space, the active space would immediately switch to shared group mode. Only when collaborative mode is explicitly requested would the space switch to the new mode.
3. Supervised mode. This mode is an alternative mode for groups, when a user may need more

permissions than the group to an activity. In this mode only the stated "supervisor" is allowed more permissions than the group. The supervisor would not obtain more permissions than his original system role is allowed.

An Active Space is dependent on the set of users in the space, and users entering and leaving the space would cause switches shown in table 2.2

Tab. 2.2: Mode Switching in an Active Space

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

The model they extend is RBAC to utilize the role concept. The objects in their model are the services in the 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.

Each active space provides services for different activities. Each instance of an activity in an active space is called a "virtual space." Each virtual space will have different access control requirements, and the access control policies and mechanism should still allow for seamless interaction while still maintaining seamless transition between virtual spaces. Their requirements for the model are:

1. It provides dynamic support for explicit cooperation between users, groups of users, and

devices.

2. Individual access rights should not leak across groups and users neither gain nor lose their rightful privileges during collaboration.
3. Physical aspects of the real world should affect security in the space.

The model they present provides support for both mandatory and discretionary access control though they use RBAC as their base. In their model, there are three types of roles: system, space, and application roles. System roles define the user's generic permissions for different accesses of object in a system. Space roles define the policy within an active space where a system administrator sets access control for resources in the space. Application roles define the access control policy for an application. When a user enters a space, the system role is mapped into a space role. Also, application roles are mapped into space roles and access control is performed on the resulting space roles. The space administrator creates the mapping between the system and application roles and space roles by examining the generic permissions that refer to real resources in the space. The mapping ensures that users are not allowed more permissions than necessary.

The architecture of the system involves an authentication service that identifies the user and issues their credentials. These credentials follow each request for service that a user makes. Here, applications run on behalf of the user and make use of the devices. These applications are on the user side. Software services provide interfaces for user applications to use the device. The space administrator sets access control policies, which are in the form of access control lists. The access control services include inceptor that intercepts all requests to services, and only permit authorized ones. The inceptor may be implemented by the service or by the space software infrastructure, transparent to users and applications. The objective is to control accesses to the services within a space, where applications use services to access devices.

2.2.3 Generalized Role-based Access Control

Covington et. al. targets a context-aware access control model for securing applications that create, manipulate, or access information about residents or resources in their homes, [14]. The Aware Home at the Georgia Institute of Technology is a pervasive infrastructure where the group's research ranges from sensor applications to security architecture. Covington *et al.*'s research focused on a context-aware access control model utilizing the features of the Aware Home. Their goal was to provide a model to support policies that make use of security-relevant environmental data to control access to resources or information manipulated by applications.

Their model, the generalized role-based access control (GRBAC) introduces the notion of environment roles. It applies the notion of roles from RBAC not only to subjects but also to system state. Each system state is represented by an environment role. They allow designers to write policies from a subject-centric, object-centric, or environment-centric point of view. Their *generalized role-based access control* (GRBAC) model removes the RBAC limitation of a subject-centric only viewpoint while still exploiting RBAC's core feature of roles to organize all entities and group environment states and objects. Access control, then, depends not only on subject roles, but also environment roles.

GRBAC extends RBAC by adding environment roles, where it is based on the state of the environment. The RBAC's notion of a role is generalized to capture the state of the environment.

It is used to apply similar properties of role activation, role hierarchies, and role separation to manage complex policies that depend on environmental state. A permission p is granted by user u if the permission, subject role, and set of environment roles is in the set of permission assignments, the subject role is active for user, u, and the environment roles that are currently active contain the set of required environment roles. Instead of applying permissions to a single role, their model assigns permissions to sets of roles, where the set may include both subject and environment roles.

Environment roles become active as environment conditions are met. Depending on what environment roles are active, the users may or may not be granted permission to their active role. These environment roles can change continuously and rapidly. Similar to subject roles in RBAC, environment roles are activated, revoked, managed through hierarchies, and prevented from conflicts through separation of duty.

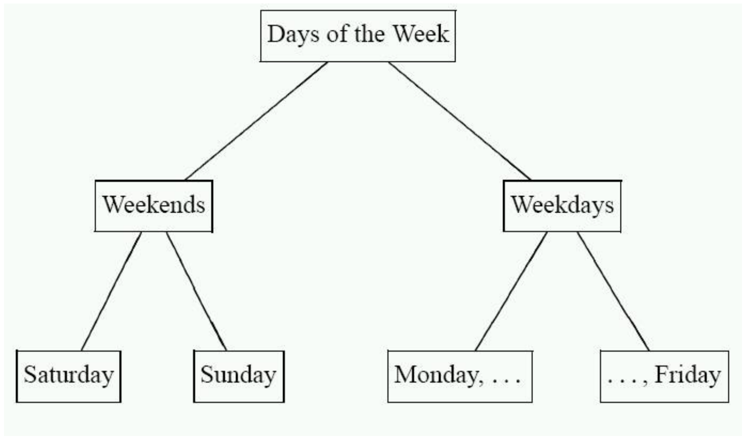

Fig. 2.3: GRBAC Environment Role Hierarchy Example

An example used in the literature is where a role would correspond to each day of the week or month of the year. Figure 2.3 shows an environment role hierarchy where each environment role represents a day of a week. The environment weekends contains *Saturday* and *Sunday*, while *Weekdays* role contains the *Monday* to *Friday* roles. Then, the *Days of the Week* role can be formulated to form *Weekends* and *Weekdays*. Environment roles could also be used to describe rules that represent subject locations, such as *upstairs*, *downstairs*, *master bedroom*, etc. By using environment roles, subjects are granted permission depending on the pattern of the user activity within the organization and the time periods in which they are expected to request access to a resource.

In addition to the RBAC formal model, GRBAC has the following components:

- ER and EC stand for environment roles and environment conditions, respectively.
- $PA \subseteq P \times R \times 2^{ER}$ captures permissions assigned to user roles when a given environment role set is not active. PA here associates a permission not only with a subject role, but makes it conditional on a set of active environment roles.
- $EC \rightarrow 2^{ER}$ represents environment role activation for a duration of a session. Condition changes require other roles to be evaluated each time, and hence, based on the environment conditions, a set of environment roles are activated at the time of request.

2.2.4 Dynamic Role-based Access Control

The context-aware dynamic access control model (DRBAC) [71], shown in Figure 2.4 is a context-aware access control model that involves access decisions that depend on the context/state of the system and the user's credentials. Since user context is highly dynamic, consideration of context into security may result in problems that may compromise the user's access privileges. DRBAC dynamically grants and adapts permissions according to the current context. In their approach, each user is assigned a role subset, and resources are assigned a permission subset for each role that will access the resource. State machines are maintained by delegated access control agents at the subject level to navigate the role subset, and the object to navigate the permissions subset for each role. The state machine consists of state variable (role, permission), which encode its state and commands, which transform its state. These state machines define the currently active role and its assigned permissions and navigate the role/permission subsets to react to changes in context.

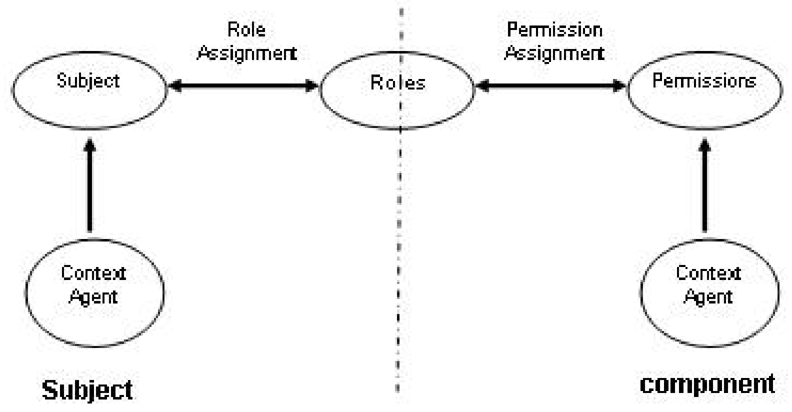

Fig. 2.4: DRBAC Model

DRBAC has two main requirements: (1) User privileges must change when the user's context changes and (2) A resource must adjust its access permissions when its system information changes. In addition to RBAC's model, DRBAC adds an additional entity, ENVS, which represents the set of context information in the system. A context agent is used to collect context information in the DRBAC system. The set of roles during a user's session will change dynamically among his assigned roles. Context information is used to decide which role is active, where a user will access the resource with the active role.

DRBAC uses a central authority to maintain the overall role hierarchy. When the user logs into a system, a subset of the role hierarchy is assigned to the user for each session. The CA then sets up an agent for the user and delegates user rights to the agent. The agent monitors the environment user status and dynamically changes the user's active roles. Each resource maintains a set of permission hierarchies for each potential role that will access the resource. The resource maintains its environment and dynamically adjusts the permissions for each role depending on the environment state. Hence, each user would have a role state machine, and each object has its own permission state machine. The context agents trigger transitions in these state machines upon collection and analysis of context information.

2.2.5 Context-Based Secure Resource Access

Tripathi *et al.* focuses on unobtrusive mobility by providing secure and transparent access to resources and services [64]. They show a way to have an environment proactively discover and transparently bind the resources required by the user, while negotiating policies set for by the users and for the resources. The issues they present are the requirement of determining a user's access privileges on a resource, access authorization based user context, visibility of private or sensitive user information in untrusted environments, whether the requested service/resources is trusted, and restriction of services/resources to guest users.

To be able to provide secure and transparent access to resources, policies must be considered from several different points: (1) the user provides a policy about his/her information and resources; (2) the resource/service manager has policies about access control from users; and (3) a policy is provided for each collaboration activity describing when and under what conditions a user is allowed access to resources. Hence, the architecture contains three views: the user view, manager view, and context view. The user view contains its policy for resource access and a list of resources it may access. The manager view contains detection policies for the context manager while it is responsible for resource changes depending on the underlying context and the resources available. The context manager detects any changes to context and notifies the view manager.

The underlying access control model extends RBAC by including reactions, resource descriptions, resource bindings, and view access control. The roles are defined in terms of activities, where an activity is an abstraction of a collaboration that provides the scope for the roles, objects, and privileges. Similar to operations, reactions are executed when a precondition is computed to be true. A reaction does not always involve a user and may run on behalf of a role owner. To succeed in allowing separate entities to communicate events are created and sent to one another through events by user applications, activity instantiation events, and physical/runtime events. Altogether, these events form the underlying communication that allows each entity to be aware of its surroundings.

2.3 Summary

We provided a background of access control models. In particular, we presented the Access Control Matrix, Discretionary Access Control, Mandatory Access Control, and Role-based Access

Control models. We also presented related work to access control models directed towards Pervasive Computing and context-awareness. In the next chapter we will provide a Pervasive Computing environment framework and present our access control model for Pervasive Computing environments.

Chapter 3

Activity-based Access Control Model

In this chapter, we present our view of a Pervasive Computing (PC) environment and our context-aware access control model. In particular, we present the meaning of context and context-awareness and specify major components of the PC environment. Then, we present the activity-based access control model, a model which adapts to the environment's context.

3.1 The Pervasive Computing Environment

Moving towards a Pervasive Computing environment perception of the physical space will surely change. A room is no longer a passive structure where one may enter and leave without anyone taking notice. When a person enters a room, information is gathered, recorded, and distributed about the person to services and applications in the environment. This information could be a person's location, services accessed by the person, and people she may be associating with. This information about the user and the user's surroundings is known as *context*. In the following subsections, we discuss the Pervasive Computing environment. Further, we discuss in detail the definition of context and the various components that make up a Pervasive Computing environment. We present our view of a PC environment through compilation of many works throughout the past decade. Many have their own understanding and view of PC environments within their specific area, but there is no one common conception of a PC environment. A PC environment is unclear and hence we describe our view through careful collection of many other PC literatures. We have formulated a PC environment through extractions and compilations of work from many different areas in Pervasive Computing.

First, we present the notion of context in Subsection 3.1.1 and 3.1.2. To understand PC applications or services, one must understand the meaning of context-aware computing. In Subsection 3.1.3 we present our view of a PC environment and its components. Also, we address a problem, which became apparent through our evaluations and would be found in any PC environment.

3.1.1 What is Context?

Context is easy to understand for people in conversations since it is an intuitive notion that refers to the surroundings of the center of interest. It provides additional information about whom, what, and where to increase situational understanding [36]. When defining context for computing technology, the task becomes more challenging, since the term is vague and has more of an intuitive connotation. A concrete definition of the term, *context*, has been discussed for some time, but it has not yet been agreed upon. The following are definitions that have been stated in works over the years.

The term “context-aware” was first introduce by Schilit and Theimer [54] in 1994. They define context as the “location of use, the collection of nearby people and objects, as well as changes to those objects overtime”. In 1997, Brown *et al.* [8] defined context as “any information that can be used to characterize the situation on an entity, where an entity can be a person, place, or physical computational object”. Context could be information such as location, identities of people around the user, the time of day, season, and temperature. In 1997, Ryan *et al.* [47] defined context as the users location, environment, identity and time. In 1998, Dey [17] defined “context as the users emotional state, focus of attention, location and orientation, date and time, objects, and people in the users environment”.

Then, in 2001, Dey gives the following definition [18]:

Definition 3.1.1.1. Context is any information that can be used to characterize the situation of an entity. An entity is a person, or object considered relevant to the interaction between a user and an application, including the user and the application themselves.

We use the final definition given by Dey in [18] since it includes both explicit context where information is given by the user through a user interface and implicit context where the information requires some computational process before it may be active. Information that does not affect the user or an application is not considered context. Context is only information related to the purpose of the current task.

3.1.2 What is Context-Aware Computing?

The first researchers to introduce context-aware computing were the Olivetti Active Badge [66] work in 1992. Since then, there were many attempts to define context-aware computing. These definitions typically fall into two categories: using context and adapting to context. Both categories require applications to detect, interpret and respond to context.

Schilit and Theimer [55] defined context-aware computing as the "ability of a mobile users applications to discover and react to changes in the environment they are situated in". The software adapts according to its "location of use, the collection of nearby people and objects, as well as changes to those objects overtime". Hull *et al.* [29] and Pascoe *et al.* [42] states that context-aware computing is the ability of computing devices to detect and sense, interpret and respond to aspects of a user's local environment and the computing devices themselves. Dey [17] limits context-awareness to the human-computer interface instead of the underlying application. Salber *et al.* [48] define context-awareness as the ability to provide maximum flexibility of a computational service based on real-time sensing of context.

Other researchers [8, 25, 55, 67] define context-aware applications as applications whose behavior changes as the user and application context changes. Ryan [47] states that context-aware applications are applications that take input from environmental sensors, users select from a range of physical and logical contexts according to their current interests or activities. Brown [8] defines context-aware applications as applications that automatically provide information and/or take actions according to the users present context as they are detected by sensors. Fickas *et al.* [22] define environment-directed, synonymous with context-aware, applications as those that monitor changes in the environment and adapt their operation according to predefined or user-defined guidelines.

We use Dey's definition of context-aware computing which states [18]:

Definition 3.1.2.1. A system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the users task.

3.1.3 Components

There are many components that make up a Pervasive Computing environment. We state the main components are administrators, users, user applications, brokers, services, and resources. Administrators of environments define the capabilities that it may provide. Services and resources are installed for users to use during the time they are present in the PC environment. They define regulations that all users, services, and brokers must abide, according to the home or organization rules. Users roam about the environment carrying out tasks and utilizing services and resources provided by the environment. User applications interact with users facilitating their requests and possibly determining user intention. When a user application receives a request or determines a user intention, it attempts to execute the task. If the task could not be accomplished by the user application with the resources on the device on which it is situated, it must request additional support from the environment. It contacts a broker from the environment to request additional services. The broker determines or provides the best choices of services available in the environment that is able to succeed the task. Through additional communication the services are contacted and help accomplish the task by reading, manipulating, or executing resources available to the service. In this subsection, we build a PC environment through evaluation of various areas in PC and present our interpretation of the infrastructure.

3.1.3.1 User Application

Definition 3.1.3.1. User applications in Pervasive Computing are context-aware applications which communicate directly with users and adapt to changes in circumstances in the environment. They provide an interface for users, interpret user requests or intentions, and execute the formalized task from the interpretation.

User applications may be any process that is on a user's peripheral. These devices would be considered mobile devices. Banavar [4] states the application is a means for a user to accomplish a task. It is not a means to exploit a device's capabilities or available services in the environment. The application would discover and identify other applications and services that are in its local vicinity. We believe it is necessary for the application to negotiate with brokers since the device and peripherals may not have all the resources needed to perform a task. Also, the application may require additional resources to split the execution since it may not have acceptable performance on a single device. Performance may be difficult on a peripheral device since it may only have limited resources. Also, our notion of a user application may require a means to communicate with the user. The device may not have an interface or enough communication resources to interact with the user. In this case, the user application must utilize other means, such as environment services and devices, to interact with the user.

The main features of a user application in Pervasive Computing are its ability to adapt to the environment context, discover available services, and interpret and deliver user objectives. User objectives may either be implicit or explicit stated. Users may directly request services which the application must discover to carry out the task. These requests are known as *explicit requests*. *Implicit requests* are requests that are interpreted by the application through received context and history of user actions and requests. When activities occurring in the environment change, the application behavior would also change based on the users' intent.

3.1.3.2 Service Broker

In this section, we present our notion of a broker. It is necessary to have a mediator between the user application and the environment services. Without a mediator, the user application will be unaware of services that are provided by the environment. Also, the services in the environment would be unaware of requests made by the user application whether they are implicit or explicit user requests. The service broker provides awareness for the entities that request its assistance. We define the service broker as follows:

Definition 3.1.3.2. A service broker is a process existing in the Pervasive Computing environment that assists user applications in accomplishing tasks from requests by querying for services and negotiating actions between the user application and the services.

Similar to applications on personal computers, brokers are applications in the PC environment which gather services and resources to formulate a way to perform a task. The service broker formulates a composite set of services that would satisfy a task. Such responsibility requires knowledge of service functionalities and coordination between each service in the composition. Therefore, the service broker must be able to discover available services [11,23,46,58].

3.1.3.3 Context Provider

There has been much research on developing a context provider [15,19,27,28,30,35,38,44,45,49,56]. The means to collect, interpret, and distribute environment context from sensors and other services is provided by the context provider. The context provider collects and maintains context from various entities such as sensors or other services. Lei *et al.* [35] addresses three critical issues relevant to the design of a context provider. These issues are the sensitivity of information aggregated and distributed, information quality, and context source participation. When a context provider distributes the information it collects, it must consider the users' privacy. The users' privacy must be protected to prevent misuse of the information. The context provider must also ensure the information collected from various sources must be accurate. It must be able to express inaccuracies and uncertainties in the collected data. A Pervasive Computing environment may have multitudes of sources for context data wanting to publish the data they discovered. These sources must send the data to the context provider so it can be processed and distributed. Unfortunately, information collection must be controlled to prevent overloading the context provider. It cannot always consider every possible source that provides context.

Lei *et al.* [35] also extends Dey's [18] definition of context to include not only transient context information but also include persistent context information. Dey's definition considers only transient context where it represents only a single instance of time, where Lei *et al.* considers persistent context, representing a recurrent pattern of transient context. For example, when Dan wakes in the morning at 10:00 a.m., this context is considered transient context information, but if he normally wakes at 10:00 a.m. the context information is persistent context information. This notion is quiet necessary for context distribution since there may be applications that need histories of context used to interpret a user's intent. For example, Dan goes to a meeting at 1:00 p.m. each day, but because of some circumstances he missed the meeting. A meeting application may receive both the transient context and the context that Dan is not present at the meeting. It would then notify Dan of his absence and send him an update of the meeting agenda and notes.

We formally define context provider as follows:

Definition 3.1.3.3. A context provider is a context service in the Pervasive Computing environment that collects, interprets, maintains, and distributes collected context information to various applications or other services that request its service.

3.1.3.4 Services

Pervasive Computing services in the environment are the feature components that support core functionality of the environment. They are the entities that aid the users in succeeding tasks when user applications are unable to carry out the task itself. The services communicate with environment devices directly to send, manipulate, or read data which may in fact change the device state. It may also read, modify, or write electronic files. We categorize these accesses into two distinct categorizations: device manipulations and electronic file accesses. Device manipulation may vary the environment state just as a person entering the room may change the environment state. Although there may be times that presence of users in a room and actions made by users may not be controlled, actions made by devices can be controlled and must be manipulated with extreme care.

It is necessary for services to be aware of the context of its local vicinity and the user's context. Services must be able to coordinate its decisions to deliver a request with the environment context. These decisions may be critical to the users' safety. For example, a dentist who is taking x-ray for a patient typically exits the room and prevents any other people from being in the room or near the doorway. When standing outside, the decision made by the service handling the x-rays must also ensure there is no person near or entering the room other than the patient. Therefore, decisions to take an x-rays depend on the context of others present.

Definition 3.1.3.4. An environment service is a service existing in the Pervasive Computing environment that may be context-aware and assists users in accomplishing tasks through its designed functionality.

3.1.3.5 Resources

Resources accessible by users consist of two types in the Pervasive Computing environment: devices and electronic files. Hardware devices installed by an administrator or owner of the environment are controlled by services embedded in the environment. In general, users will no longer access and control them directly or though some controls attached in the room, but they are able to read, manipulate, or write data directed to the resources from their user application. Relevant information about these resources must be available for other entities to browse and access.

Definition 3.1.3.5. Resources are devices or electronic files existing in the Pervasive Computing system.

3.1.4 The Environment Space

A PC environment space consists of four components: the physical space, the user space, the virtual space, and the resource space. Altogether, these entities interact with one another to achieve goals based on the user's intent.

Definition 3.1.4.1. An environment space is defined as a collection of users, services, and resources in continuous time.

A user application is an application that interacts directly with the user and may retrieve context about a specific user to provide an interface to services in the environment through environment brokers. It may either be on a personal device or embedded into the environment space. It processes commands interpreted from the user interaction. Note the communication method is not necessarily always involving a display screen with text or pictures, a keyboard, or a mouse. It is left to PC hardware and software designers to settle the interaction mechanism between the user and the user application. The user application interacts with the user, receives direct or indirect requests of service, and executes controls to deliver the service. It communicates with brokers in the PC environment when resources are not locally available or when it is incapable of performing a task by itself. If it requires services that it is unable to deliver itself, it may request services from the environment space. The request typically involves some resource, either data or device, which some environment service may be able to deliver.

We separated processes that aggregate various services to accomplish a single task and that are self-contained (that is, a process that does not require communication with other services in the space and does not directly communicate with user applications). The first type is an *environment broker* and the latter is an *environment service*. These processes exist from installation to their removal time in the space. The environment brokers listen for requests and perform actions to carry out requested tasks by sending further requests to environment services. The environment brokers check whether the user is allowed to perform the request and performs the required functions depending on their privileges. The environment services act as a mediator between the brokers and the resource controller. It receives requests from the brokers and communicates directly to the resource controller to access or modify the resources' state.

These environment brokers may asynchronously service multiple users in parallel instances, during existence of activities in the space of which it is responsible. The environment brokers check user privileges and perform their request. The services only perform the requests as they are received without regard to consistency in the environment. This process creates problems with seamless interactions between the user and environment. A second request to change the state of a resource may override an early request that would violate the first user's intentions without their expectation. Thus, constraints must be applied to requesters of resource state changes and involvement of context constraints is the objective of this research.

Allowing for requests that are independent of the environment state brings forth a consistency problem. Formally, the consistency problem is defined as follows:

Given a Pervasive Computing environment with services and applications that receive automated, implicit, or explicit requests, privilege misuse must be prevented and activity objectives and intentions must be preserved.

Due to high mobility and concurrent services possible in an environment space, issues arise when services are unaware of the environment state causing goals and intentions to collide. For example, suppose users in an environment have the privileges to control information received by the projector service to display information on a large screen during a meeting. (1) How is user β prevented from sending information to be displayed when user α is supposed to be the only person presenting? (2) How does one prevent an external subject from interfering with the current environment settings? (3) If the presentation information was proprietary, how is data prevented from being retrieved and sent to an external user? (4) How does one prevent user β from messing with the lights, air, or peripherals, such as a printer, while the presentation is occurring?

Based on the goal of seamless interactions and unobtrusiveness, user applications, environment brokers, and environment services adapt to the space context, inferring user intent when possible. User applications and services will avoid direct communication with users to be nonintrusive, but this may cause disruption in user tasks. User interaction avoidance, context adaptations, and user intent inferences may cause more problems than intended. When inferring intent and adapting to spatial and temporal context, decisions made by user applications may not always be what the user intended. Conflicts with intent inference become obtrusive since the result may require users to fix the cause of the problem, present a distraction in a user's current activity, or risk user safety. It becomes possible that users, user applications, and services may make decisions that risk the systems security and user safety. The distinction between users, user applications, environment brokers, and environment services is required to control each entity's access requirement. The goal of this distinction is to allow the possibility of containing brokers to its designed functionality. Brokers act on behalf of the users, but are contained within the assigned permissions that limit allowed services.

Activities occurring in the space define the environments state, and the present users, the active brokers, the active services, and the current context define the current activity. Users are represented by their roles in an activity. In a presentation example, a role would either be a presenter or presentee. The intended activity is a meeting where there is only one person presenting while the others are listening and interacting directly with the presenter. Some services present in the environment are controllers for the lights, projector, sound, door, and access control and authentication service. These services are embedded in the environment and interact with environment brokers which are similar to applications aggregating services to perform a task. For example, a presenter wanting to present information on the display screen requires the projector service and the light service (to dim the light intensity so the display is viewable). Hence, the presentation broker must be utilized to achieve the presenter's task. In the environment space, there are also basic services that a typical user may control such as the lights and door. To be able to achieve a seamless interaction between the user and the environment, the components in the system must abide by some set of rules to preserve integrity in the pervasive system. They must be aware of the environment state, constraints that follow the state, and rules of action required upon grant or denial.

3.2 Activity-based Access Control Model

The vision of Pervasive Computing changes security requirements for subjects participating and utilizing the system features and capabilities. Unless each subject is allowed to access, manipulate, and write to any data of and device, an access control mechanism is needed to guarantee the confidentiality and integrity properties of security. Confidentiality is the assurance that information access is limited and only allowed by authorized users, where integrity refers to the trustworthiness of the informational resources, assuring the resource has not been modified inappropriately.

The goals of access control are much more than just determining whether a user has access to resources. When a user wishes to perform some operation on some object, the user would need to make a request for such action. The system determines from the policy associated with the user whether the authorization for such action exists in the policy. So, based on the policy contents and existence of the authorization for the user, the system will either grant or deny the user's request. Many models have been developed and used to design the system with those needs such as the access control matrix, mandatory access control, discretionary access control, and role-based access control model discussed earlier. These models have the intent of protecting resource access based solely on a user's status in an organization. Therefore, policies contain access privileges for any given user. Role-based access control differs from other access control mechanisms since their policies are focused on the user role and not the individual. Although it is possible RBAC may implement the previous mechanisms [6, 40, 51], its core feature is still the notion of a role, and it allows for separation of duty. Although we would like to allow for permission or role delegation, the area is undergoing research under RBAC and has not been fully agreed upon. Hence, we will leave this area out of this research until the RBAC delegation research has stabilized.

The goal of this research is not to provide an enhanced model for current systems, but its essential purpose is to provide a model for Pervasive Computing environments, specifically, a context-aware access control model. Pervasive Computing environments differ from current systems because access decisions may depend on the context in which the requests are made [1, 14, 24, 70, 71].

3.2.1 Are traditional access control models enough?

Due to Pervasive Computing requirements and capabilities, traditional models are not sufficient to provide the awareness needed to protect the users and resources. We consider traditional access control models to be models such as MAC, DAC, and RBAC. These models do not allow additional information other than set privileges to be used in access control decisions. RBAC allows the additional feature of separation of duty, but the information is based on prescribed role, providing mutual exclusion between various roles upon assignment or activation.

Decisions of access control in PC environments require the mechanism be context-aware. Although applications and services are aware of context such as its requesting user's location, activity, and surroundings, they may be unaware of appropriate behavior and settings when allowing access. Typically, with traditional access, these applications and services would only check whether a user requesting some action was assigned the permission. If the user has the right to perform the action, then the service or application will perform the request. In current physical environments, it is easy to see whether some user is accessing a physical object, but in the PC world, these accesses will be invisible and preventing such accesses would be much more difficult.

Definition 3.2.1.1. A change to the physical environment state caused by a subject access request, where changes are noticeable by present users, is known as a *physical environment state transition* (PEST).

A PEST may affect every user in the environment. While we would like to provide nonintrusiveness and transparency as a PC goal, PESTs may cause distractions and disruptions during the user's tasks. Tasks that are to be accomplished must respect the tasks that are active. There is a time to perform certain tasks, but all applications and services must coordinate with one another and the user activities to be able to efficiently and transparently accomplish current tasks.

Apart from intrusive PESTs that must be avoided, context-awareness may be required in some activities. For example, Alice was cooking dinner, and she received a phone call that her child was hurt at school. In her distraught mode, she grabs her car keys and drives to her child. Unfortunately, she forgot to turn off the stove and oven. A context constraint is required to prevent any fire hazards. In this case, the context constraint would specify that all hazardous electrical kitchen appliances are allowed if and only if the person cooking is present in the home. The system would detect that Alice was no longer within the proximity of her home, and automatically turn of the oven and stove after some given time interval.

It may be possible for the oven and stove to be aware that Alice's presence is no longer there, but if every context-aware device has a different policy for different context changes, the entire PC system would not be coordinated and may then be intrusive to the current event. Hence, an access control system where all rules and regulations are stored and accesses are determined is required to coordinate the services and brokers.

3.2.2 Access Control Requirements

Requirements for access control must be stated to determine the goals of our access control model. In this section, we present a compilation of access control requirements from related work and add an additional requirement.

Requirement 3.2.2.1. The security service must be "ubiquitous", nonintrusive, and transparent [1].

The requirement of "ubiquitous", nonintrusive, and transparent services is an obvious goal since this requirement stems from the origin of Pervasive Computing. The access control service in collaboration with the user applications must be able to grant, revoke access rights, and activate/deactivate user status without the intrusion of user interaction. The interaction with users must be limited until there is some critical or essential response required.

Authentication services must also be nonintrusive. In this thesis, authentication is assumed to be existent in the system to enable subject identification. There is a large research field for authentication such as biometric authentication or context-aware authentication [5, 7, 13].

Requirement 3.2.2.2. The security mechanism must provide multilevel protection. It must be able to provide different levels of security based on the security policy, environmental activities, and available resources [1].

Usually, access to resources is not treated equally because of sensitivity and ownership of the resource. Hence, multilevel security is the concept of having different classifications and categories for the resource that would allow access to users only with security clearances. Classification level is the sensitivity level that is associated with some information. Multilevel security is necessary to prevent classified information from uncleared users.

This requirement also captures the effect of applying context constraints in access control. Since access control will depend much on environment context and upon violation of the context constraints, precaution and prevention measures will occur to prevent user accesses. The fluctuation of dynamic access privileges based on environment context provides multilevel protection to protect information and users in different scenarios.

Requirement 3.2.2.3. The security system must support a security policy language. The policy definitions must be descriptive, well-defined, and flexible. The language must incorporate rich context information and provide physical security awareness.

A security policy language must exist for the security designer to express the organization's rules and regulations so the security system can enforce the specified policy. The policy must provide enough well-defined description to encapsulate all the designers rules. Since the security system is context-aware and accesses depend on context, security policy language must be able to describe context constraints. This language must then be flexible and incorporate rich context information and allow for physical security awareness.

Requirement 3.2.2.4. The access control mechanism must be able to change user privileges dynamically based on context information [14, 71].

Similar to requirement 3.2.2.2, this requirement explicitly states that the access control must dynamically and automatically modify user privileges when certain context changes. A user is assigned some set of permissions. At any point in time, only a subset of those permissions will be active. Based on active roles and environment context, the set of active permissions will become active for the user.

Requirement 3.2.2.5. The access control mechanism must dynamically support explicit cooperation between different users, groups of users, and devices [50].

When users are collaborating and working in groups, different levels of access privileges are required for the group. The access control mechanism must be able to handle group permissions to allow for a shared space. Typically, a room is shared by multiple groups for different activities at different times. Each of these groups will have different access control requirements in the shared space. Hence, the security policy must be able to capture the shared space and the different groups. Adaptations of the role-based access control provides collaboration control of privileges [2, 9, 12, 24, 26, 41, 57, 60, 63, 65].

Requirement 3.2.2.6. The model must ensure that a user's access privileges do not "leak" across groups [50].

When allowing for collaboration between groups and managing group privileges, the privileges of a user must not be transferred to an unauthorized user. Each group in a shared space will have different levels of security requirements and access. Any leakage of privileges between groups would be a security violation. All active privileges should remain only in the group in which it was initiated.

Requirement 3.2.2.7. The users must neither gain nor lose their rightful privileges during collaboration [50].

When allowing for collaboration, the users must not get additional privileges nor should they get fewer privileges than they are allowed. Each user has the right to use the privileges that were assigned to them. The system and the users are not allowed shared permissions unless the permission is assigned to all the users. The point of access control is to allow control of accesses only to authorized users. All permissions were given to users based on their role in the organization. Any leakage of privileges between groups is a violation of the policy.

Requirement 3.2.2.8. The security system must protect against intrusive PESTs, physical environment state transitions, where a state transition is any sensible change which is easily noticed by users.

The system must be aware of each access requests the user makes. Although the user's permissions may explicitly allow users to perform a certain action, the action may cause disruption and distractions in the space. PESTs are unavoidable and necessary, but decisions for PESTs made by a service must be carefully controlled and user safety must always be a consideration in critical cases.

3.2.3 Access Control Model Requirements and Assessment Criteria

Tolone *et al.* [63] provided access control model requirements and assessment criteria during their collaborative systems research. The assessment and criteria in their work fits the purpose of our goal as well as the requirements in the previous section. In PC environments all active entities **must** collaborate with one another, whether they are associated with each other or not. From a high-level, any entity could cause a PEST, and the ability to control PESTs requires awareness and collaboration between groups. From a user level of collaboration, users will be working either individually or in groups, support for collaboration is necessary to allow users to communicate and cooperate on common tasks.

3.2.3.1 Requirements

We present Tolone *et al.*'s [63] requirements of an access control model for collaborative systems. By the nature of Pervasive Computing, the system is a collaborative system requiring all entities, users, brokers, services, devices, and user applications to collaborate with one another when activities are similar or even dissimilar. Cooperation is required between each entity. Additionally, users will either work individually or in groups. Collaboration is required in such cases to allow information and resource sharing, but the sharing of information and resources requires careful control and proper authorization.

Requirement 3.2.3.1. Access control must be applied and enforced at a distributed platform level.

Requirement 3.2.3.2. Access control models should be generic and enable access rights to be configured to meet the needs of a wide variety of cooperative tasks and enterprise models

Requirement 3.2.3.3. Access control for collaboration requires greater scalability in terms of the quantity of operations than traditional single user models.

Requirement 3.2.3.4. Access control models must be able to protect information and resources of any type and at varying levels of granularity.

Requirement 3.2.3.5. Access control models must facilitate transparent access for authorized users and complete exclusion of unauthorized users in a flexible manner that does not constrain collaboration.

Requirement 3.2.3.6. Access control models must allow high-level specification of access rights, thereby better managing the increased complexity that collaboration introduces.

Requirement 3.2.3.7. Access control models for collaboration must be dynamic. It should be possible to specify and change policies at runtime depending on the environment or collaboration dynamics.

Requirement 3.2.3.8. Performance and resource costs should be kept within acceptable bounds.

3.2.3.2 Assessment Criteria

In this section we present Tolone *et al.*'s assessment criteria. We believe many of the specified criteria encompass the criteria of an access control for Pervasive Computing environments.

Criteria 3.2.3.9. An access control model should not be overly complex since it may lead to unforeseen problems and implementation can become difficult.

Criteria 3.2.3.10. Understandability is a core requirement since it defines the model's transparency and its underlying principles.

Criteria 3.2.3.11. Ease of use indicates how simple the system is from the end user's point of view in terms of its usage. If it is cumbersome to use, then there is a chance that users will not favor it.

Criteria 3.2.3.12. Applicability of an access control model is an indication of its practicality.

Criteria 3.2.3.13. Support for collaboration is the most important aspect of consideration for access control models devised for collaborative environments. Following are the specific aspects that determine the ultimate usability of any access control model.

1. Groups of Users
2. Policy Specifications
3. Policy Enforcement
4. Fine-Grained Control
5. Active / Passive
6. Contextual Information

3.2.4 Role-based Access Control Extension

Since the development of the role-based access control model, variants of the model have been developed to support features that RBAC did not consider. RBAC has its own directed audience, and proposed models have goals that are directed in different directions. RBAC's goals, as an alternative to mandatory access control and discretionary access control, was to enforce enterprise-specific security policies [20]. In many enterprises, the users do not "own" the information that they access in comparison to discretionary access control. In these cases, the corporation is the "owner" of the system resources. RBAC allows access decisions to be based on the users' roles that are part of the organization. Hence, the notions of roles are the core feature that role-based access control presents. Access permissions are associated with roles, and users are made members of the appropriate roles.

Role-based access control has limitations to security goals in PC systems regarding access control. It concentrates on user control to protect resources. Although not of its purpose, it is not able to regard the context or environment state when making decisions of access control during an instance of a session. We first define RBAC entities in terms of pervasive environments as follows:

1. *users* are people who request services in the system.
2. *brokers* are applications in an environment space acting on behalf of the user or the environment to accomplish a task.
3. *subjects* represent users or brokers.
4. *roles* are collections of permissions in order for a subject to accomplish a specific task.
5. *permissions* are authorizations to perform some operation on an object.
6. *operations* are the actions the subjects are able to perform.
7. *objects* are resources (files or peripherals) of which operations are acted upon.

8. *sessions* are instances of communication between the user and system. In Pervasive Computing, a session is created when the first subject or user declares her presence in an active space, due to the system's capability of presence awareness.

The model consists of sets of 6 elements called SUBJECTS, ROLES, OBJECTS, OPERATIONS, SESSIONS, and ACTIVITY. SUBJECTS are assigned to roles and roles are assigned permissions. A role is a way to name a many-to-many relationship among subjects and permissions [21]. A session is a mapping between a subject and activated subset of roles assigned to a subject. An activity is a mapping between subject sessions and roles.

A user is a human being while a broker is an autonomous agent. A role is a job function within the context of an organization associating some authority and responsibility with the subject assigned the role. Permission is an approval to perform an operation on one or more RBAC objects. An operation is an executable image of a program, which upon invocation executes some function for the subject. The types of operation and objects that RBAC controls are dependent on the type of system in which it will be implemented [21].

The essential goal is to protect system resources. When applying RBAC to a Pervasive Computing environment, we are protecting objects, where an object is an entity that contains or receives information. An object can represent information containers and system resources such as a printer. The set of objects are the objects listed in the permissions that are assigned to roles.

A role is a semantic construct for formulating a policy [21]. There are the subject assignments and the permission assignment, formulating a many-to-many relationship between subjects and roles and roles and permissions. This prevents subjects from being assigned more access than needed to a resource due to the limited control over the type of access that can be associated with subjects and resources.

A session is a mapping of a subject to one or many roles and a single activity. A subject establishes a session during which the subject activates some subset of roles that she or she is assigned. Each session is associated with a subject and each subject is associated with one or more sessions [21]. Each session is associated with one activity, and each activity is associated with one or more sessions.

An activity is a mapping of sessions and broker roles. After a subject establishes a session, the session is associated with an activity. An activity may include one or more sessions. The user requests brokers to join activities they are participating in.

In the following subsections, we further describe our definition of subjects, and roles. We, also, explain dynamic separation of duty since we incorporate the notion into our model. Then we present a new entity called an activity, and how sessions are affected by the entity.

3.2.4.1 Subjects

A subject, defined in Section 2.1.1, is an active entity, generally in the form of a person, process, or device, which causes information to flow among objects or changes the system state. Although the definition states that a subject could be a person, process, or device, most access control mechanisms only consider the subject to be a person, and hence, the NIST RBAC model [21] only considers users (persons) as the entity and user processes would act on behalf of the user.

The scenario in Pervasive Computing is different in the sense that it may not be the users that are requesting access. Allowing the ability for user applications to determine user intent and having service brokers request and execute actions through communication with other services, the access of information may be vigorous, overloading systems and confusing decisions made by the brokers and services. Additionally, the ability for brokers to request support from any service that is provided may indeed be dangerous and vulnerable to attacks which give the ability for unauthorized users to obtain access when they are not allowed. Since the behavior of a user applications cannot be constrained when it is located on the user's device, we constrain actions of request at the broker level, and therefore we require brokers to be an entity in the access control system. The purpose of including brokers in the model is to constrain its functionality and allow the ability to monitor and control its actions. Each broker in the environment has a single task, and it is to aggregate and execute services that accomplish user goals requested by the user application. Users are associated with roles that are defined as their position in their organization, but all permissions associated with these roles may not be needed for the current task. When requesting a broker to help accomplish a task, the broker role may be limited to only services it is associated with. The collection of permissions active at any point in time for a single broker would be much more limited than the permissions for a single user. This would prevent brokers from accessing unauthorized information when some vulnerability, attack, or malfunction is probable.

3.2.4.2 Roles

Roles may be assigned to members in homes, such as the parent and child role. Further roles could be defined within a family to consider a mother, father, younger sibling, and older sibling, or even the family's puppy. We use the concept of roles in our model to express the position the user has within a system.

Roles assigned to brokers are also a sufficient specification of its allowed task. Brokers are unique only by the functionality that they provide, and brokers in various environments or even in the same environment may overlap in the tasks they support.

3.2.4.3 Separation of Duty

Additionally, one of the most desired features of RBAC is the ability to enforce separation of duty [31,52]. The purpose of separation of duty rules is to prevent a single user from accomplishing all parts a task when the task requires two or more users. This is to prevent collusion or fraud. Kuhn provides a safety condition in [31] that must be met that would ensure the separation environment. This is stated as follows:

> Let $C[t] : task \rightarrow 2^{privilege}$ be a mapping from tasks requiring separation of duty to sets of privileges needed for those tasks. Then to ensure that no one person can perform all parts of a task t, no user can have access to all privileges in C[t]. The safety condition for separation of duty is:

$$(\forall u)(\forall t)(\forall i) : (C[t] \not\subseteq \bigcup_{i \in role | u \in M[i]} P[i])$$

Since in our model we consider dynamic separation of duty, where users may be assigned roles that are mutually exclusive but allowed to be active in one role at a time, we shall present the definition of dynamic separation of duty [21]:

Definition 3.2.4.1. $DSD \subseteq (2^{ROLES} \times N)$ is collection of pairs (rs, n) in Dynamic Separation of Duty, where each rs is a role set and n is a natural number ≥ 2, with the property that no subject may activate n or more roles from the set rs in each $dsd \in DSD$. Formally:

$\forall rs \in 2^{ROLES}, n \in N, (rs, n) \in DSD \Rightarrow n \geq n,$

and

$\forall s \in SESSIONS, \forall rs \in 2^{ROLES}, \forall role\_subset \in 2^{ROLES}, \forall n \in N,$

$(rs, n) \in DSD, role\_subset \subseteq rs, roles\_subset \subseteq session\_roles(s) \Rightarrow |role\_subset| < n$

3.2.4.4 Activities

In this section we discuss the extension of the role-based access control model that allows for environment control, collaborations and context-awareness. Access control is applied to systems to control subject accesses to resources. Subjects are typically considered to be users. In any environment a user takes advantage of his assigned permissions to accomplish some task, and users may work in groups or teams to carry out a goal. A task is a specific piece of work required to succeed a goal. Each task requires some set of permissions for it to be accomplished. Consideration of goal-oriented access control tends to be too fine-grained to provide a general access control model to support a generic Pervasive Computing environment. Task-based access control [39, 61, 62] limits contexts that are related to activities, tasks, or workflow progress and is implemented by keeping track of usage and validity of permissions [63]. Collaborations cannot easily be divided into tasks. On the other hand, team-based access control [60] or context-based team-based access control [24] do not consider rich collaborative contexts.

The context-aware access control model by Covington *et al.* [14], called the Generalized Role-based Access Control model, applies the notion of environment roles to capture environmental state. Similar to user roles, environmental roles possess the features of role activation, role hierarchy, and role separation. Instead of applying permissions to individual roles, permissions are assigned to sets of roles, both subject and environment. Problems with environment roles, as stated by Covington *et al.*, is the system administrator is required to define each environmental variable, conditions and variable values, in which the environment role will be active. Another problem is the number of environmental roles that is active at any point in time. The number of active environment roles cannot be controlled and is unknown during access time. Active environment roles may be both relevant and irrelevant to access decisions. Additionally, the environmental roles are very fine-grained. Environment roles represent environmental states, such as weekdays (Monday, Tuesday, etc.) Requiring fine-grained definitions of environmental roles becomes overly complex and the activation and deactivation of each of these roles may easily overload the system. In terms of access control models, the terminology and concept of environment roles is counterintuitive [3].

In our model, we present the *activity* entity to capture role, context, and situational mutual exclusion requirements. An activity is conceptually recognized by end-users, designers, and administrators. In terms of our model, an activity is an entity that is associated with a set of roles and a set of context constraints. Formally,

Definition 3.2.4.2. An activity is a collection of tasks specified by the capabilities of its associated roles and constrained by the context of the environment and its participants.

We can apply the goal-oriented design of context constraints. Strembeck *et al.* [37, 59] provides a goal-oriented technique to apply context constraints. Although they apply context constraints to permissions, the idea could be applied to an activity. We considered four different constraints: role-cardinality, activity separation, activity context constraint, and role context constraint. Further details are explained in Section 3.3.

3.2.4.5 Sessions

In RBAC, users activate roles through sessions. A session is a dynamic feature that associates a user with some set of activated roles. In our model, we further extend this feature to associate each session to an activity. Each session may only be associated with a single activity, but an activity may be associated with multiple sessions. In an active state, activities are associated with a set of active users whom have activated some set of roles to be associated with the activity. In the base model, an activity only limits the roles that may join the activity.

3.2.4.6 Summary of Basic ABAC

We summarize our model in the following definition.

- SUBJECTS, ROLES, OPERATIONS, OBJECTS, SESSIONS, AND ACTIVITY
- SA $\subseteq$ SUBJECTS $\times$ ROLES, a many-to-many mapping subject-to-role assignment relation

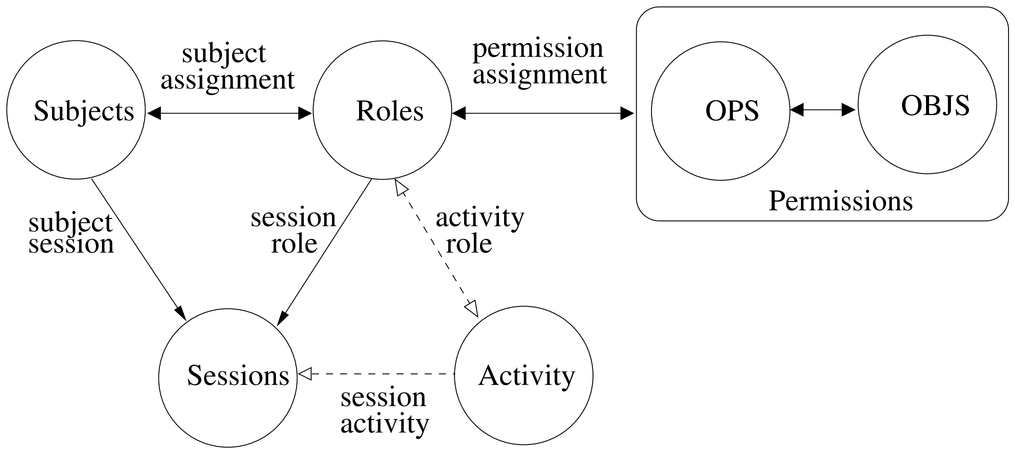

Fig. 3.1: Activity-based Access Control Core Model

- assigned_subjects: (r:ROLES) $\mapsto 2^{SUBJECTS}$, the mapping of role r onto a set of subjects.
- PERMISSIONS = $2^{(OPERATIONS \times OBJECTS)}$, the set of permissions
- PA $\subseteq$ PERMISSIONS $\times$ ROLES, a many-to-many mapping permission-to-role assignment relation
- assigned_permissions(r:ROLES) $\mapsto 2^{PERMISSIONS}$, the mapping of role r onto a set of permissions
- subject_sessions(s:SUBJECTS) $\mapsto 2^{SESSIONS}$, the mapping of subject s onto a set of sessions.
- session_roles(s:SESSIONS) $\mapsto 2^{ROLES}$, the mapping of session s onto a set of roles.
- activity_roles(a:ACTIVITY) $\mapsto 2^{ROLES}$, the mapping of activity a onto a set of roles
- activity_sessions(a:ACTIVITY) $\mapsto 2^{SESSIONS}$, the mapping of activity a onto a set of sessions

The access control model extends the RBAC model by involving activity entities that apply constraints on an active session to preserve system integrity and environment broker roles that handles the execution of services in an environment space and the validation of user privileges through the access controller. The access control considers both the credentials of the authorized users and system constraints that limit the activities in the environment. The users are assigned user roles while the environment brokers are assigned environment broker roles. To contrast, users are human beings while environment brokers are processes in the environment that serve the purpose of handling user access grants, denials and execution of requests which involve services provided by the environment.

3.3 Activity Constraint Model

The main purpose of constraints is to allow for higher level restrictions in security policies. In RBAC, there are two main categorizations of constraints, static and dynamic.

Definition 3.3.1. Static constraints are constraints whose evaluation is done during role assignment to users, otherwise known as administration time.

Definition 3.3.2. Dynamic constraints are constraints whose evaluation is done at run-time. The constraint variable values are based on the current activations within the system.

The focus of this research is on dynamic constraints, where each activation and deactivation of roles and activities depend on the dynamic environment context and the access control system.

We present two types of constraints correlating with activity activation, maintenance, and deactivation. Two additional constraints correlate with the role constraint which handles role activation, maintenance, and deactivation apart from dynamic separation of duty for activated roles.

Constrained ABAC adds constraints to the activating roles and activities. It adds separation of activity relations to enforce conflict of interest policies to prevent an activity from exceeding a reasonable authority for its purpose. Individuals of different skills or divergent interests are required to participate in separate tasks. Additionally, some tasks require the presence of individuals with different skills or authority. This is to ensure that activities are limited to individuals of specified skills or authority. ABAC allows for constraints on role and activity activations and maintenance.

3.3.1 Role-Cardinality Activation Constraint

Conflict of interest in ABAC may occur when a user engages with other users where their skills or interests are divergent. This does not mean that users may not be associated with other users of different skill code or interest. This means the information shared within an activity is restricted only to authorized users, not necessarily requiring users to be associated with the same role. Separation of duty on activated role in an activity is to enforce the constraint on the assignment of users to activities. Such a constraint would require an activity, such as a parent-teacher meeting, to have a minimum of 1 parent and 1 teacher role activated before the activity could activate. The *activity role cardinality* is defined as follows:

Definition 3.3.1.1. $ARC \subseteq ((ACTIVITY \times ROLES) \times N)$ is a collection of pairs (ar, n) in the Activity Role Constraint, where ar is an activity role, and n is a natural number ≥ 1, with the property that no more than n roles may be activated in activity a.

3.3.2 Activity Separation

Within a given environment or setting, conflicts of interest occur when activities are occurring simultaneously or too many activities of the same type are concurrently occurring. Mutually exclusive activities (MEA) allows activities to be authorized when two or more activities do not create a conflict of interest when acted on independently, but produce policy concerns when activated simultaneously. For example, without this constraint, the activities, lecture and personal internet relay chat, may occur at the same time. By allowing for MEA, activities may be prevented from activation and prevent contradicting activities from occurring simultaneously. MEA is defined below:

Definition 3.3.2.1. $MEA \subseteq (2^{ACTIVITY} \times N)$ is a collection of pairs (as, n) in Activity Activation Constraints, where each as is an activity set and n is a natural number ≥ 2, with the property that no activities may have n or more activations.

3.3.3 Context Constraint

We provide two types of context constraints: activity context constraint and role context constraint. Context conditions are applied to both activities and roles associated with the activity. For an activity or role to be active, all context conditions associated with the entity must be true. Associated with each condition is a set a context variable that must be active and ready to be validated. The context constraint validation only occurs if the role requirements, role-cardinality activation constraint, and activity separation have been validated and satisfied. Upon a session's association with an activity, only then are the associated contexts variables subscribed to and received from the context provider. Hence, an active view of all contexts is not required, but only relevant context to the activity is supplied. Users requesting to take part in an activity are allowed to utilize the permissions only when all validations have been satisfied.

Throughout the duration of an activity, revocations of activities and roles may occur based on environment and user context. A generalization required in the context constraint language is that any constraint may not be directed towards a specific user. Since we are taking advantage of RBAC's concept of roles, we must retain that advantage. Besides, specifying constraints directed towards each user is inefficient and difficult to maintain. Instead, constraints are directed towards the roles associated with an activity. In turn, users associated with the role and activity will be associated with the constraint. Violations of activity context constraints will *deactivate* the activity, and violations of role context constraints will *deactivate* those roles associated with the activity. Notifications and time-outs are required before deactivation to avoid abrupt actions from occurring unless the violation is critical.

3.3.3.1 Activity

To authorize a user the system must determine the request source. Since Pervasive Computing environments are information-rich in context, this information is used to verify the environment conditions of the participating users and brokers to ensure the purpose of the activity. Through context constraints, activities are limited to the context conditions specified in a policy.

Definition 3.3.3.1. $ACC \subseteq (ACTIVITY \times CONSTRAINTS)$ is a collection of pairs (a, c) in Context Constraints, where each a is an activity and each c is a constraint, with the property that activity a is activated if and only if constraint c is satisfied.

3.3.3.2 Roles

As well as activity constraints, we separate context constraints for activities as a whole and each individual participant. Activity context constraints deactivate an activity if any of its constraints are not satisfied, while activity role constraints deactivate only subject roles. This prevents participation of only specified role when those constraints are not satisfied. This avoids deactivation of an activity when it is the case that a single role does not satisfy the constraints; and without the active role the activity still satisfies the constraints.

Definition 3.3.3.2. $ARCC \subseteq ((ACTIVITY \times ROLES) \times CONSTRAINTS)$ is a collection of pairs (ar, c) in Context Constraints, where each ar is an activity role and each c is a constraint, with the property that role r in activity a is activated if and only if constraint c is satisfied.

3.4 Discussion

The activity-based access control provides many of the specified requirements. We evaluate our model according to the criteria specified by Tolone *et al.* [63]. Table 3.1 shows an assessment of a few collaborations: access matrix model, role-based access control model, task-based access control, team-based access, context-based team-based access control, and Covington's context-aware access control using environmental roles.

Tab. 3.1: Characterization of Access Control Models for Collaborative Systems [1]

Criteria	Matrix	RBAC	TBAC	TMAC	C-TMAC	Env. Roles
Complexity	Low	Medium	Medium	Medium	Medium	High
Understandability	Simple	Simple	Simple	Simple	Simple	Simple
Ease of Use	Medium	High	Medium	High	High	High
Applicability	Medium	High	Medium	Medium	High	High
Groups of users	Low	Y	Y	Y	Y	Y
Policy Specification	Low	Y	Low	Y	Y	Y
Policy Enforcement	Low	Y	Low	Y	Y	Y
Fine grained control	N	Low	Low	Y	Y	Y
Active/passive	Passive	Passive	Active	Active	Active	Active
Contextual info.	N	Low	Medium	Medium	Medium*	Medium*

[1] Provided by Tolone *et al.* in Access Control in Collaborative Systems [63]

We will assess our access control model and compare against these evaluations. Since our model's foundation is the role-based access control model, our model will be at least as good as RBAC. We show our evaluation in Table 3.2

With complexity, we believe the degree is equivalent to RBAC, TBAC, TMAC, and C-TMAC and not as high as Env. Roles since our added feature of an activity simplify context constraints and collaborations within an environment. The counterintuitive environment and specification requirements of each environment state in Covingtons model cause the model to be more complex than required. The understandability of an activity hides the additional activity constraints from designers, but maintains the underlying principles of the model. In any case, our model requires the user to indicate their activity. This concept is not difficult to specify. The abstraction of an

..

Criteria	ABAC
Complexity	Medium
Understandability	Simple
Ease of Use	High
Applicability	High
Groups of users	Y
Policy Specification	Y
Policy Enforcement	Y
Fine grained control	Y
Active/passive	Active
Contextual info.	Medium*

Tab. 3.2: Activity-based Access Control Assessment

activity encapsulates multiple tasks, and, hence, the user is required only to specify the general activity she needs to accomplish his task. The complexity of use is mainly due to the role activations required, which we inherited from RBAC. Our model is practical for general purposes. The flexibility presented by the additional constraints we developed allows our model to be used by many different environments.

In addition, we inherit RBAC's ability to support groups of users, policy specification, and policy enforcement. Our access control model allow fine-grained policies since we are able to control the activation and activities users participate in based on role constraints, activity separation, and context constraints. Additionally, our model is highly dynamic since activations and deactivation of roles and activities depend highly on other active roles, activities, and the environment context. The degree to which we require contextual information is medium. At a minimum, our model possesses the capabilities of RBAC, but at a maximum, our model may be fully dynamic if all constraints are applied to each activity and role.

3.5 Summary

The methodologies we present protect users from being hindered by the system and prevent unauthorized accesses by introducing the notion of activities. Activities are tasks that are restricted or allowed based on the context conditions collected in the environment.

The activity and role-based access control model is based on brokers, roles, users, sessions, activities, and permissions. The users and brokers are constrained by the roles they were assigned which are assigned a set of permissions. Additional to the model, 4 constraints are presented: Activity Role Cardinality, Mutually Exclusive Activities, Activity Context Constraints, and Activity Role Context Constraints. Activity role cardinality constrains the number of roles that may be activated and associated with an activity. The mutually exclusive activities prevent activation of two or more activities that contradict one another. Activity context constraints allow for context constraints to be applied to activities, where activities are activated and remain activated if and only if all assigned constraints are satisfies. The same concept is applied to activity role context constraints except the constraints are applied to the roles associated with an activity; and the role is prevented from activation when the associated constraints are not satisfied.

Chapter 4

Design & Implementation

The activity-based access control is a model, which extends the role-based access control model to provide context-awareness in security designs and control. The key concept in this model is the notion of activities. The addition of the activity entity enhances the PC environment to provide a safe, consistent, and cooperative environment. Activities specify roles, environment context, and other activated activities related to current tasks. In this chapter, we will design, implement, and demonstrate an access control system conforming to our access control model. We provide requirements needed for the system, an architectural model, design of system components, and a system demonstration.

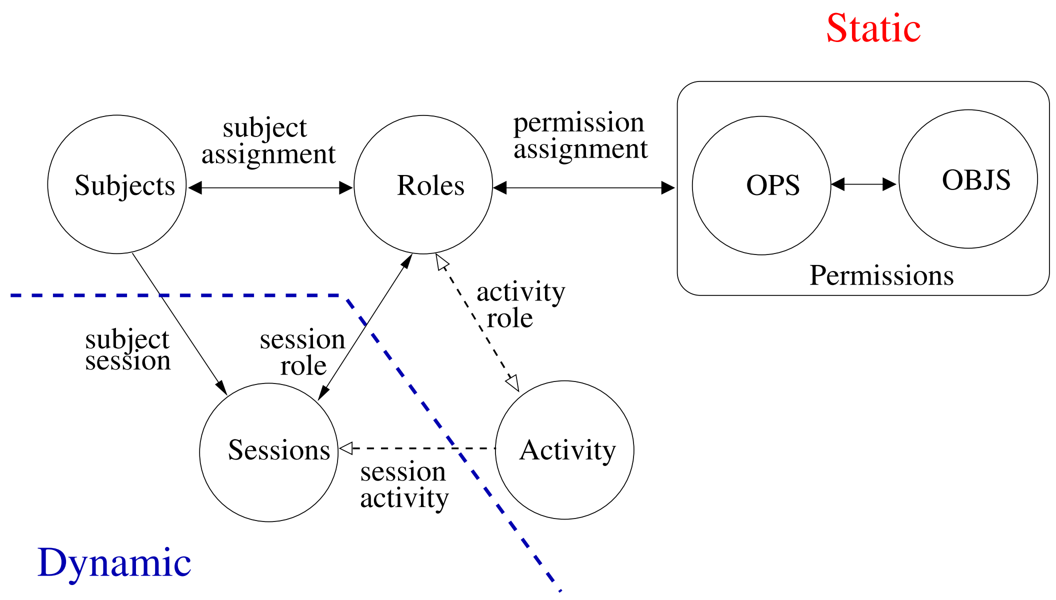

Fig. 4.1: ABAC Model

4.1 Features

To break up the model into subproblems, we first divide the entities and relationships into static and the dynamic portions, as shown in Figure 4.1. The static portion represents entities and relationships related to policy definitions, while the dynamic portion is related to subject activations and deactivation. Policy definitions are created by system administrators. These policies are created before the access control system is deployed and may evolve over time. Once the system has been deployed, subjects may become involved with the system through activations and deactivations of activities and roles. This is the dynamic portion of Figure 4.1. In this chapter, we will exclude RBAC features since its specification is available from the NIST foundation [21]. We will specify modifications we have made and concentrate on the ABAC features. Below, we present features to capture the scope of the model, which we will design and implement.

1. ABAC entity and relationship definitions. The essence of the ABAC model is the ability to define an activity and the roles contained within the activity. In this feature, administrators are given the ability to define activities supported by the system and their associated roles. During subject interaction with the system, sessions and sessions_roles are created. In addition, an activity is associated with each session to ensure that only roles specified by the administrator are allowed to participate in the activity.

2. Role Cardinality Requirement Specification. Feature 1 specifies roles allowed to participate in an activity. The role cardinality requirement specifies roles required during participation. This requirement specifies which roles are required for activity activation and roles are restricted from activation. In addition, the feature explicitly states the minimum role cardinality required before activity activation and the maximum number of activated roles allowed during an instance of the activity.

3. Mutually Exclusive Activities. Conflicting activities may occur when not restricted. This feature is used to protect the user's safety and objectives. Administrators define activity sets to prevent conflicting activities from occurring in a single environment.

4. Activity Context Constraints. Activity context constraints are used to ensure that activity activations occur according to the environment context. Administrators define context constraints that correlate with an activity through the availability of context information provided by a context provider. Subjects who wish to activate an activity must conform to the rules specified by the administrator.

5. Role Context Constraints. Similar to Feature 4, the role context constraints place constraints

on roles participating within an activity. Role activation and participation must satisfy each of these specified context rules. When rules are associated with the role, the context information is translated to the context of the activating user. Hence, context constraints are generalized for the role taking part in the activity, but in actuality, the rules requiring role context retrieve the context of the activating subject. Additionally, activation and participation may involve context that is not of the activating subject but other subjects and context information. This feature allows context restrictions for individual roles, which would cause deactivation of roles and not the entire activity.

Our features at a minimum are as secure as the Role-based Access Control unless a role-based system is not chosen. To take advantage of ABAC, Feature 1 must be implemented to allow for activities. The other features provide constraints allow consistency in the environment. We are aware that additional constraints will present overhead in the systems' resources, but the balance between providing consistency, security, and safety needs to be weighed. In comparison to many other PC access control systems, we moved much of the access control decisions away from user devices since user devices may be limited in resources. Many user peripherals may not be capable of the computing or storage required for access decisions, such as storing system or role states. The access decisions are made by our system, which will be embedded in the Pervasive Computing environment.

The four constraint features we provided are independent of one another, and hence a security designer may decide to choose any combination for their system. A security designer may require only role cardinality constraints or mutually exclusive activities. There is one case where two of these feature may be combined, and that is between Feature 2 and 4. The access control system may be an entity that participates in providing context to the context provider. In the process of providing context information, an information loop back may occur resulting in receipt of interpretations of current activities or roles. The access control system may provide to the context provider, information of activity activations, role activations, and participating subjects. This allows the system to determine activity role cardinalities in the system and context rules relating role cardinalities may be applied.

In the next section, we provide requirements for our ABAC implementation incorporating its features.

4.2 Requirements

Essential to any software design are the software requirements. In this section we present requirements for both the static and dynamic portions of the model. The static portion is known as the policy definition. This involves definition of entities, relationships, and constraints, which will be used specifically by system administrator. A mechanism is required for system administrators to specify system policies. The dynamic portion involves the dynamic system functionalities during subject and service interaction. Since we will implement all parts of the ABAC model, we provide requirements encapsulating each of its features.

4.2.1 Policy Definition

Policy definitions involve the definition of the static portion of the ABAC model shown in Figure 4.1. This section provides requirements for policy definition. The following are specifications required by an administrator:

1. Entities: activity
2. Role cardinality bounded constraints: minimum and maximum cardinality activity role participation
3. Mutually exclusive activity set
4. Context constraint support entities: context constraints
5. Activity revocation criticality
6. Activity revocation notification and time-outs

An administrator first creates an activity, such as an office meeting. An activity may represent a task or group of task within a home or organization. Allowed subjects are then determined for this activity. This allows related roles to be grouped with one another for joint collaboration. This allows the system to realize subjects that are within the same activity. Mutually exclusive activity sets provides a mechanism to prevent activity collisions and preserves user safety and intents. Context constraints allow the system to be aware of its users and prevent activities from occurring out of context. An issue of activity and role deactivations occurs when constraints are not satisfied. Should the role or activity be deactivated immediately, or should it have a holdoff time to allow an application to end gracefully? We provide three additional variables that control an activities deactivation methods, which are defined by an administrator. The activity revocation criticality specifies whether an activity should be deactivated immediately. Critical activities are deactivated instantly. Otherwise, an activity revocation may be incurred by warnings and time-outs. This allows subjects to react to the changing situation and prevent deactivations. Individual role revocations occur only when its context constraints are no longer satisfied. In such a case, two events may occur. When a role context constraint is no longer satisfied, its deactivation may cause a role cardinality constraint conflict. In such a case, the activity itself would undergo revocation, but only if the role has been deactivated. The sessions containing this role would undergo revocation, and if its deactivation causes a conflict, the entire activity undergoes revocation. The other case is when a role is not mandatory and not restricted from joining the activity. In such cases, violations of role context constraints would only cause the role to undergo revocation, and the activity would not be affected.

4.2.2 Dynamic Subject Interactions

Subjects interact with the PC environment to utilize its services. The ability to utilize services requires a subject to have permission for the corresponding action. Our system notifies requesters and receivers of the subject's access rights, and notifies them of access grants, denials, and activations. Before privilege requests are allowed, subjects must undergo role and activity activations. Activity activations occur when a session containing roles become associated with an activity. During this time, context information related to the activity is collected, and constraints are checked.

An activity will undergo revocation when a constraint is violated after the activity has been invoked. Depending on the criticality of the activity, the activity or role may be revoked from the subjects immediately or after a specified number of warnings. In our case, we specify criticality as a Boolean value. If an activity's criticality value is true, a session's participation is deactivated immediately if any constraints are violated. Otherwise, the time-out, t, and notification count, n, is used during revocation. The subject would be notified n times with t milliseconds separation. In this subsection, we specify requirements for four main functionalities: activity activation, deactivation, revocation, and context validation.

Activity Activation and Deactivation Activity activation occurs before a session is associated with any activity. The subject must request to join the activity before a subject may make use of their privileges. Additionally, a subject must be able to deactivate a role or a session from an activity. The following specifies requirements for activity activation and deactivation.

1. A session must be associated with an activity before the session is active.
2. An active activity must contain only roles specified by the administrator.
3. Activated roles must be within the activity's cardinality bounds.
4. Activity activations and participations must satisfy the activity's context rules.
5. Role activations and participations must satisfy the role's context rules.
6. There must not exist any mutually exclusive set of active activities.
7. A subject must be able to disassociate his role from an activity.
8. A subject must be able to disassociate his session from an activity.

Activity/Role Revocation and Context Validation Constraint checking requires activity and role revocation. The subject must be notified if constraints are being violated, but the amount of time before an activity is deactivated depends highly on the criticality of the activity. For example, an activity where a meeting is extended beyond its time lot with no other critical event is inconsequential, such as a classroom lecture. On the other hand, it is critical to deactivate visiting hours activities in a hospital when a patient's pulse rate has become below a certain threshold. Below are requirements for activity revocation, role revocation, and context validation.

1. For non-critical activities, all subjects must be notified before revocation.
2. For critical activities, the activity must be revoked immediately and subject must be notified.
3. All contexts in rules associated with an active activity constraint must be subscribed to the context provider.
4. All active constraints must be checked upon receipt of context with differing values.
5. If an active condition is not satisfied, all active activity or roles associated with the condition must undergo notification and revocation.

4.3 Functional Specification

Section 4.2 provided general requirements for the Orion implementation. In this section, we provide functional specifications for administration, activation, deactivation, revocation, and constraint checking. These specifications follow our ABAC model and its entity and relationship naming convention.

4.3.1 Policy Definition

Our implementation provides activity definition, activity associations with roles, and activity mutual exclusion. Below are specifications which an administrator must be able to add and delete.

- Entities: $activity$
- Relationships: $activity\_roles$
- Activity Mutual Exclusion: $dsaset$, $dsaset\_activities$

In the base ABAC model, activities are defined by their associated roles. These role associations are defined by the $activity\_role$ set. Each $activity\_role$ set contains minimum and maximum cardinality variables to specify the number of a specified role that must be activated. Mutually exclusive activities prevent conflicting activities from occurring simultaneously. Given an $activity$, we may specify a set of activities that may not occur simultaneously. This is represented by the $dsaset$ and $dsaset\_activities.$ $dsaset$ represents the dynamic separation of activity set, and $dsaset\_activities$ represent the relationship between $dsasets$ and $activities.$ Additionally, a cardinality, n is specified for each $dsaset.$ This cardinality specifies a mutual exclusion count, representing no more than n activities in the set may occur at the same time. As well as the mutual exclusion constraint, our model allows administrators to specify role participation requirements, notifications, and time-outs. Below are variables an administrator must be able to set to obtain these features:

- activity_role minimum cardinality
- activity_role maximum cardinality

- revocation notification count
- revocation notification time-out

Context Constraints Context constraint is a feature that requires additional entities above those provided in the model. Below specifies all entities and relationship correlating to context constraints, and assignment of activities and activity_roles to these context constraints.

- Entities: $context$, $context\_type$, $context\_subject$, $context\_constraints$, $context\_conditions$
- Relationships: $activity\_constraints$, $role\_constraint$, $constraint\_conditions$, $condition\_context\_subject$

Supporting context information must be preknown before context constraints are made. Additionally, a context value always corresponds to a context and a subject, where a subject may be any person, place, thing, or object. For example, a $context\_subject$ may be "Room 320", while the $subject\_type$ may be "office_room". An activity may be associated with a $context\_constraint$ through the $activity\_constraint$ relationship. Similarly, an $activity\_role$ may be associated with a $context\_constraint$ through the $role\_constraint$ relationship. Below is the constraint language we created to apply context constraints. The language is an adaptation of the language used by Gaia, [13, 43, 44]. Each $context\_constraint$ is a collection of context conditions formed by the $constraint\_condition$ relationship. To realize the types of context and subjects associated with a condition, a $condition\_context\_subject$ relationship is formed to correlate the three entities. The constraint language is a subset of predicate logic allowing for a single instance of a quantifier. A single instance of a quantifier means there are no embedded quantifiers, where multiple quantifiers do not exist in a single statement, or condition.

Constraint Language

condition	:=	quantifier (*subject_type*, *subject_variable*, (boolean_expr)) | (boolean_expr)
boolean_expr	:=	boolean_op(bool_expr, bool_expr) | predicate | NOT (boolean_expr)
predicate	:=	(context_term relation constant) | (context_term relation context_term)
context_term	:=	context(*context_name*, *subject_name*) |
		context(*context_name*, *subject_variable*) |
		context(*context_name*, *role_name*) |
boolean_op	:=	AND | OR
quantifier	:=	ALL | EXIST
relation	:=	< | <= | > | >= | = | <>

The constraint language allows for quantification of any subject_type. Predicates of a rule or condition consist of two types of comparisons, two context_value comparisons or a context_value and absolute value comparison. Comparisons we allow are $<, <=, >, >=, =, <>$. These predicates together may be joined to formulate propositional logic with the AND and OR Boolean operations. Quantifiers are added to formulate a predicate logic formula. Three examples of a context condition are:

Example 4.3.1.1.

```
((context('number_people', 'room_320') = '1')
```

Example 4.3.1.2.

```
all('room', x,
(and((context('temperature', x) = '70F'),
      (context('room\_activity', x) = 'empty')))
```

Example 4.3.1.3.

```
all('role', 'consultant',
     ((context('location', 'consultant') = 'room_320'))
```

Example 4.3.1.1 enforces there must be only one person in room_320. Example 4.3.1.2 exemplifies quantifiers and Boolean logic. It states that *for all rooms, the temperature of each room must be 70F and there should be no activity in the room.* Example 4.3.1.3 shows how quantifiers of a role type are user. It states that: *for all roles of type consultant, the consultant's location must be in room 320.*

4.3.2 Dynamic Subject Interactions

The access control system's dynamism involves activation, deactivation, revocation, and constraint checking, where each of these require the interaction between the systems, subjects, and services.

Activations and deactivation involves an initiation by the subject, while revocation involves subject notifications. Constraint checking is initiated either through activation, deactivation, or upon context retrieval.

Activation/Deactivation To be able to use a PC environment's services, a subject must be able to add and delete the following:

- Entity: $session$
- Relationship: $subject\_sessions$, $session\_roles$, $session\_activity$

These relationships and entity formulations are essential to the ABAC model. A session creation is first initiated by the subject when it wishes to join an activity. It would associate roles with the session and then request to join an activity. Role to session associations are represented by $session\_roles$, and subject to session associations are represented by $subject\_sessions$. Upon request to join an activity, the system must check if the new join violates any constraints. It must check the role cardinality constraints and subscribe to any context related to the activity. Upon receipt of these contexts, the system checks context constraints and either allows or denies the activity join. While the session is associated with an activity, all activity and role constraints must be satisfied, otherwise, it will undergo revocation.

Revocation When decisions to revoke an activity or role are finalized, the system must be able to delete the following:

- $session\_role$
- $session\_activity$

These decisions are invoked after role-cardinality, activity mutual exclusion, context activity and role constraints checking. Failure to uphold the system policy results in revocation. Before a revocation, the criticality of the activity is checked. If the activity is determined as critical, the activity must be deactivated immediately; otherwise, the subject is notified within a certain time-out specified by the administrator.

4.4 Activity-based Access Control Implementation

Our implementation is built requiring many components. To be able to demonstrate our implementation of ABAC, we must be able to simulate a PC environment, providing user applications, brokers, context providers, and services. Since our goal is to demonstrate our access control model, we provide a simulation of these additional entities. These entity simulations are implemented with Java, although they may be implemented using any other language. This implementation provides an interface to our access control system. It simulates the environment by allowing session creation, role and activity activations/deactivations, access checks, context updates, and access control notification receipts, such as context subscribes or revocations. In the following sections, we present an implementation of our access control model.

Figure 4.2 specifies the system's architecture. An interface to the access control system is needed for brokers and services to communicate. This system specifies four types of access control services that provide an interface to subjects and other services, presented as follows:

1. Activation & Deactivation Service. This service provides an interface for subjects to create

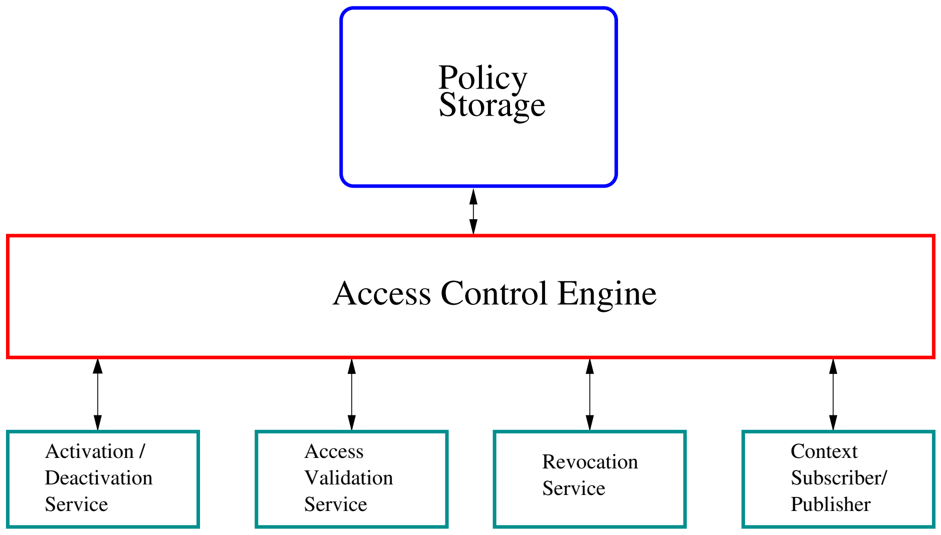
Policy Storage
Access Control Engine
Activation / Deactivation Service
Access Validation Service
Revocation Service
Context Subscriber/ Publisher

Fig. 4.2: Orion Architecture

sessions and activate roles and activities.

2. Access Check Service. The service allows services and subjects to check subject privileges. Subjects are notified only of grants and denials of their access requests.
3. Revocation Service. This is a reporting service which notifies subjects when they are being revoked from an activity or role.
4. Context Subscriber/Publisher. This service interfaces with the context provider, where context data is subscribed, received and published.

The policy storage used in conjunction with the access control engine is a critical element to access control. It provides a place for policies to be recorded and used by different environments. Orion's data storage was implemented using Postgres, while the access control engine was implemented using a mixture of Java and PL/pgSQL. Use of a database was chosen based on its ease and capability to retrieve, process, and store information. We used Postgres in particular since it does not require much administration and it provides capabilities, such as stored procedures, views, and an interface to Java. The Java portion of the implementation mainly provides a communicating interface for administrators, subjects, other services, and the context provider. The access control engine core functionality is described in the following sections.

4.4.1 Activation & Deactivation

During role and activity activation there are four possible states that either the activity or session may be allowed. These states are: *INACTIVE*, *PENDING*, *ACTIVE*, or *SUSPENDED*. An activity, session, and role state diagram is shown in Figure 4.3, 4.4, and 4.5 indicating specified stated when a constraint is satisfied of not. The INACTIVE state occurs either when an activity contains no session participations or when a session has not specified an activity to participate. A session also remains INACTIVE when a session requests to join an activity that is mutually exclusive to current invoked activities. The PENDING state for a session occurs when it has requested to join an activity, but is waiting for context related to the activity to be subscribed and received. Once context has been received, the system validates the context constraints and checks whether there are enough roles to invoke the activity. If the session context constraints were evaluated and satisfied, the session's state becomes ACTIVE; the session's state is changed to INACTIVE. If the activity's context constraints are violated or there is an insufficient number of activated roles, the activity's state stays PENDING; otherwise, its state will change to ACTIVE. For an activity to remain in ACTIVE mode, it must be associated with at least one session.

While an activity is ACTIVE, sessions may join or resign from the activity. Furthermore, a session may activate additional roles or deactivate current roles. Each of these events must be handled with special care. Note that if at any time an activity's context constraints or role cardinality constraints become unsatisfied, the activity and all its associated sessions will immediately undergo revocation. Now consider the case when a session wishes to join an invoked activity. In this case, a new session must maintain an activity's state. Initially, the system must check if the join will violate any role cardinality constraints. If this is the case, the session is denied from joining the activity. Otherwise, context subscriptions are required to validate the session's context constraints and any activity constraints that involve the new session. If there are no required context subscriptions, the session immediately becomes ACTIVE. If there are pending context constraints to be validated, the session is marked as PENDING. Once context has been received, context constraints are validated. If any context constraints are violated, the session again becomes INACTIVE and all subscribed context related to the session are unsubscribed. If these context constraints are satisfied, the session joins the activity, and it is marked as ACTIVE.

In the case when an activity has already been invoked, a session, s, has joined the activity, and s wishes to add a new role, role cardinality constraints and context constraints must be checked. If the addition of the new role violates the maximum roles allowed for an activity, the role is denied from joining. If this constraint is satisfied, context information related to this new role must be subscribed. At this point, the $session\_role$ is added, but marked as PENDING. Once context has been received and context constraints have been validated, the role addition is allowed or denied activation depending on whether they were satisfied or not. If constraints were satisfied, the role is marked as ACTIVE. In any case, if a session or session_role is removed, all related context is unsubscribed and the cardinality constraints must be rechecked. Revocation occurs when the cardinality constraints are not satisfied. If an activity or role must undergo revocation, the subject must be notified. Once an activity or role has been deemed as a violation to policy, the activity session or sessions are marked as suspended, and the subject are notified. A critical activity will be revoked immediately.

4.4.2 System Simulation & Evaluation

In this section, we provide two scenarios to demonstrate our access control system. The first scenario takes place in a business office, and the second scenario takes place in a home. We will demonstrate the role cardinality constraint, mutually exclusive activities, and context constraints. In our demonstration, we will refer to our entire environment simulation system as Orion.

4.4.3 Private Meetings

> *Susan is a consultant who is having a private phone meeting with a client, Bob, in her office, hoping to pursue a contract with them to develop a new home design. Suddenly a co-worker rushed into her room telling her about a problem with a contract they were trying to get, not realizing that she was on the phone. By the organization's policy, a private phone meeting consists only of a consultant and a client. Any violations to this policy may risk the privacy of the client or the business.*

This scenario assumes that both Susan and Bob have been authenticated with Orion to ensure that their identities are who they say they are. Also, Orion must know who Susan and Bob are to both authenticate and allow access. The environment we are simulating is Susan's office, where Orion controls information coming in and out of her office. Initially, Orion's access control system must contain a policy for users, roles, activities, and constraints. The script for these definitions is shown below (Comments in the script start with a #):

```
# generate some users
ADD USER Bob
ADD USER Susan

# generate some roles
ADD ROLE consultant
ADD ROLE client

# generate some activities
ADD ACTIVITY private_phone_meeting

# add roles to the activity allowing only 1 active role of each type
ADD ACTIVITYROLE private_phone_meeting client 1 1
ADD ACTIVITYROLE private_phone_meeting consultant 1 1

# generate some constraints
```

```
ADD CONSTRAINT persons_in_room


# generate some conditions

ADD CONDITION single_person ((context(‘number_people’, ’room_320’) = ‘1’)

ADD CONDITION consultant_location all(‘role’, ‘consultant’,

((context(’location’, ’consultant’) = ‘room_320’))


# associate conditions to constraints

ADD CONSTRAINTCONDITION persons_in_room single_person

ADD CONSTRAINTCONDITION persons_in_room consultant_location


# associate constraint to activity

ADD ACTIVITYCONSTRAINT private_phone_meeting persons_in_room
```

Now that we have defined a policy for our users, Bob and Susan, the system must realize its available context and conditions the context constraints are associated with. We define our contexts, subjects, and associations below. Although subjects do sometimes associate with subjects, for simplicity, we repeated subject additions as context subjects.

```
# generate some context

ADD CONTEXT number_people
ADD CONTEXT location


# generate some subject type
ADD SUBJECTTYPE room
ADD SUBJECTTYPE user
ADD SUBJECTTYPE role


# generate some subject
ADD SUBJECT room_320 room
ADD SUBJECT client role
ADD SUBJECT customer role
ADD SUBJECT Bob user
ADD SUBJECT Susan user
```

Next we associate constraints with context and subjects.

```
# associate conditions to context_subjects
ADD CONDITIONCONTEXTSUBJECT single_person number_people room_320 room
ADD CONDITIONCONTEXTSUBJECT consultant_location location client role
ADD CONDITIONCONTEXTSUBJECT consultant_location location customer role
```

This command indicates to add an association between conditions, context and subjects, given a condition, context, context subject, and context subject type. The context subject type is necessary to indicate the type of subject whether we are dealing with. In the following, we deal with activations and deactivations of sessions, roles, and activities. First, Bob and Susan are required to contact a video chat broker who will handle most of these activations. The broker will create two sessions, client_chat and customer_chat, for Bob and Susan, respectively. Upon creating the sessions, it activates Susans consultant role in her session and Bob's client role in his session. Their sessions are requested to join the private_phone_meeting. Next, context for the two are subscribed, and validated. The system first validates Susan's context constraints. Although her context constraints are satisfied, the role cardinality constraints require Bob's association with the activity. The system then validates Bob's context constraints, checks the role cardinality constraints, and activates the activity for both users. The following script demonstrates the activation:

```
# create some sessions
ADD SESSION Susan consultant_chat
ADD SESSION Bob client_chat


# activate session roles
ACTIVATE ROLE Susan consultant_chat client
ACTIVATE ROLE Bob client_chat consultant
```

```
# associate sessions to activity: join activity request

ADD SESSIONACTIVITY private_phone_meeting consultant_chat Susan

ADD SESSIONACTIVITY private_phone_meeting client_chat Bob


# activate context receive

ACTIVATE CONTEXT number_people room_320 room_320 room

ACTIVATE CONTEXT location Susan client role


# send initial context to access control system

UPDATE CONTEXT number_people room_320 1

UPDATE CONTEXT location Susan room_320
```

We have two additional processes in the system that subscribe to context and notify users of deactivations. When the request to join an activity occurred, our context subscriber service was notified and displayed:

```
location        Susan

number_people   room_320
```

This simulates the ability to realize which context is required and convert the client and consultant roles to specifically indicate a user. Then, we simulated a constraint violation where the subject is notified of revocations. This process handles all revocations including timeouts and notifications. In the next process, we updated Susans location to indicate that Susans coworker has entered the room.

```
# change context
UPDATE CONTEXT number_people room_320 2
```

Upon this update, our access control system receives the context and validates the context associated with the activity. Since the context constraint is violated, all sessions under the private_phone_meeting undergo revocation and the revocation process is notified of the following:

```
private_phone_meeting   client_chat     Bob
private_phone_meeting   consultant_chat Susan
```

It is notified of which activity, session, and subject that is undergoing revocation. The entire script is given in Appendix A. A full script of this scenario is shown in Appendix A.

4.4.4 Rated Movies

This scenario takes place in a home consisting of a mother, father, and child. The policy for this scenario is in Appendix B.

Jack and Jill are both parents of their daughter Mary, and they will be watching a movie together. Mary gets to the living room first and attempts to start a movie. Orion

> *detects the movie is Restricted and children under 17 are not allowed to watch without an adult. When Jack and Jill enter the living room ready to watch the movie, the movie is then started.*

In this scenario, we demonstrate how our access control system could be used in a home to allow parents to restrict their children from watching movies that are rated R or above. The full script for this scenario is shown in Appendix B. In our policy, we created a user entity for each family member and gave them either the adult or child role. We add a permission to watch a movie and created a "rated_r_television" activity. Constraints that were added state there must be an adult present in the living room for this activity to be activated. We then simulated the scenario by initially having the child create a session and request to join the activity, but due to lack of an adult in the room, the child is prevented from watching the movie. It is not until at least one adult has joined the activity the child is allowed to watch the movie.

4.5 Summary

In this chapter, we have shown the requirements and design of our access control system. The access control system implements our Activity-based Access Control model providing for role-cardinality constraints, mutually exclusive activity constraints, and context constraints. Also, we have demonstrated the system with two scenarios where intrusion occurs during a private meeting and prevention of rated R movies for children.

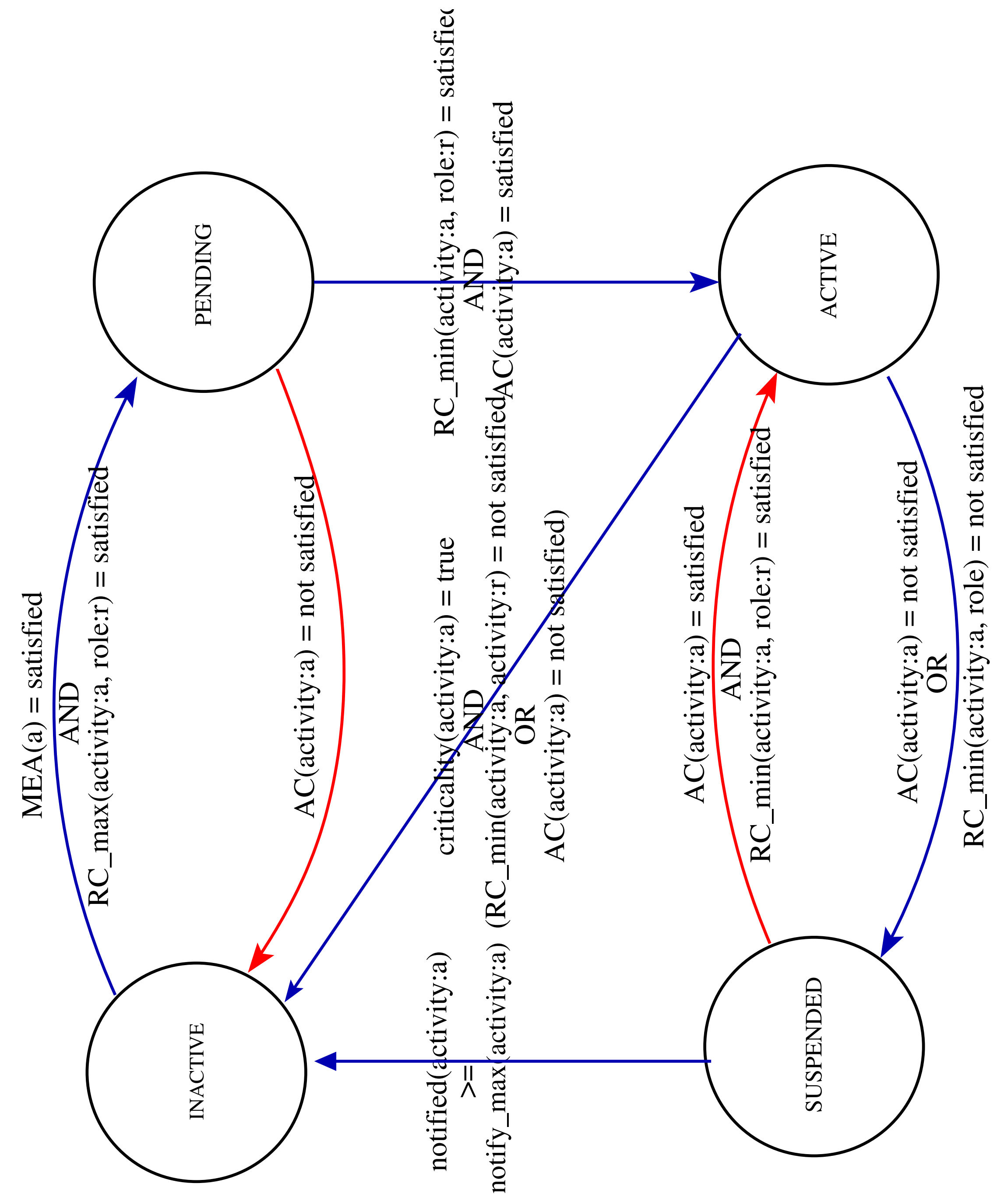
PENDING
ACTIVE
SUSPENDED
INACTIVE
RC_min(activity:a, role:r) = satisfie
AND
AC(activity:a) = satisfied
AC(activity:a) = not satisfied
MEA(a) = satisfied
AND
RC_max(activity:a, role:r) = satisfied
criticality(activity:a) = true
AND
(RC_min(activity:a, activity:r) = not satisfied
OR
AC(activity:a) = not satisfied)
AC(activity:a) = satisfied
AND
RC_min(activity:a, role:r) = satisfied
AC(activity:a) = not satisfied
OR
RC_min(activity:a, role) = not satisfied
notified(activity:a)
>=
notify_max(activity:a)

Fig. 4.3: Activity State Diagram

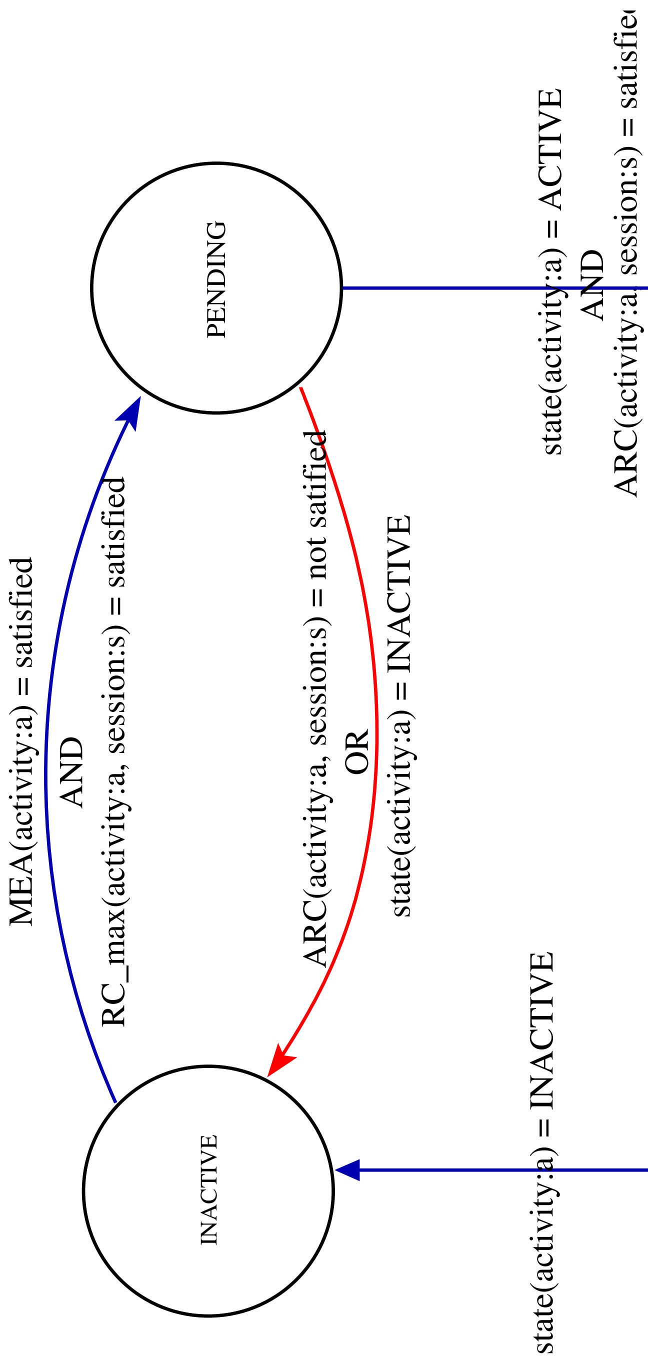
PENDING
ACTIVE
SUSPENDED
INACTIVE
state(activity:a) = ACTIVE
AND
ARC(activity:a, session:s) = satisfie
state(activity:a) = SUSPENDED
state(activity:a) = ACTIVE
state(activity:a) = INACTIVE
ARC(activity:a, session:s) = not satified
OR
state(activity:a) = INACTIVE
MEA(activity:a) = satisfied
AND
RC_max(activity:a, session:s) = satisfied

Fig. 4.4: Session State Diagram

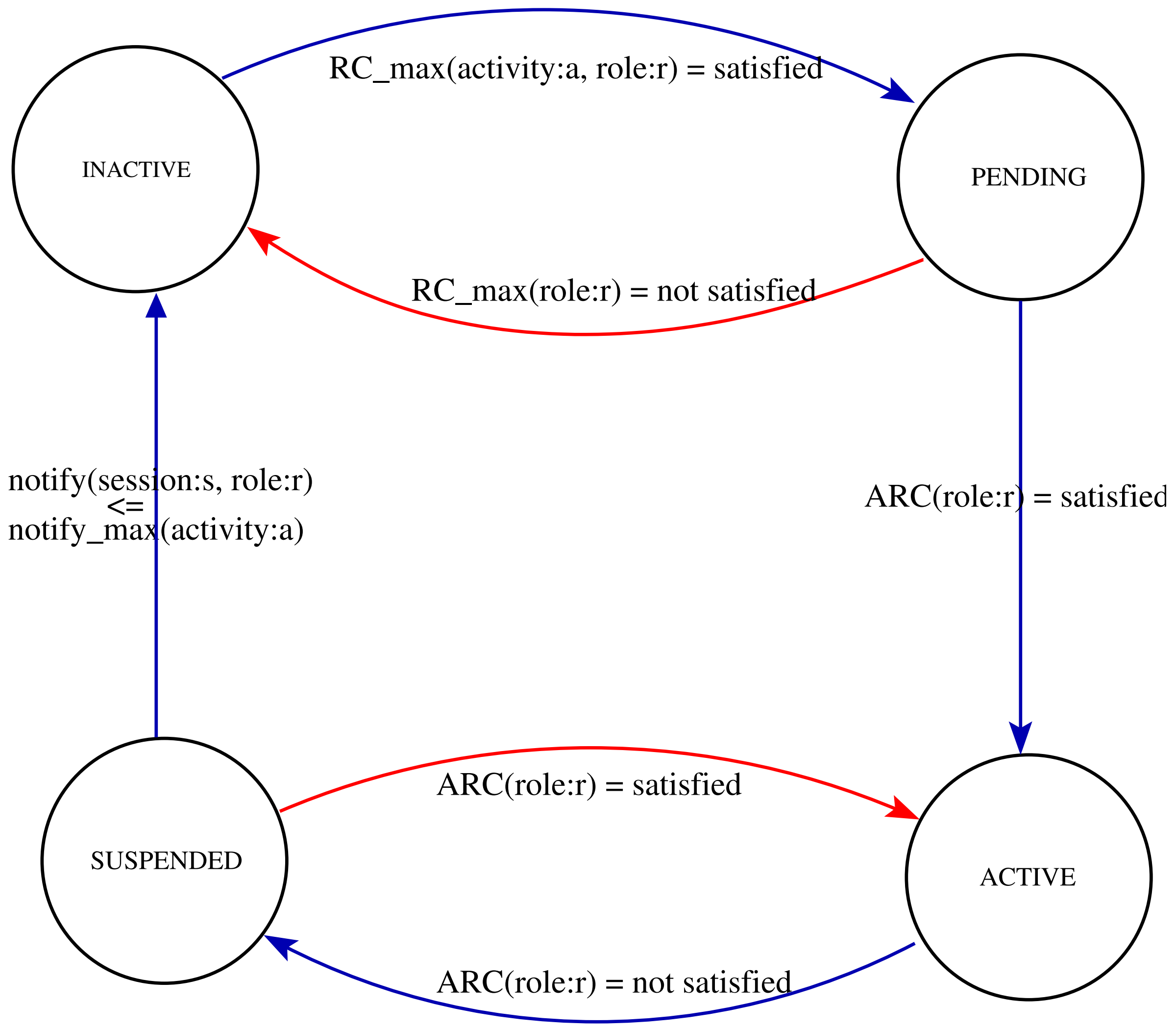
INACTIVE
PENDING
SUSPENDED
ACTIVE
RC_max(activity:a, role:r) = satisfied
RC_max(role:r) = not satisfied
notify(session:s, role:r)
<=
notify_max(activity:a)
ARC(role:r) = satisfied
ARC(role:r) = satisfied
ARC(role:r) = not satisfied

Fig. 4.5: Session Role State Diagram

Chapter 5

Conclusion

The thesis we presented describes our Activity-based Access Control model and presented our description of a Pervasive Computing environment. In this chapter, we will summarize our research contributions and describe some future work.

5.1 Research Summary

Pervasive Computing is a growing area requiring much research and organization. An essential requirement of Pervasive Computing is a universal framework for environment spaces. In this thesis, we formulated a framework from our own ideas and many different works. Through extensive research, we explained context, context-aware computing, and components of a Pervasive Computing environment.

Additionally, access control is a major component for successful deployment of a pervasive environment to protect confidentiality and integrity of the system. We also presented a new problem, which we call the consistency problem. In general, this problem states that through all the automated processes and intent inferences in the environment, violations of user intentions may occur because of the parallel processing and events that may occur. To meet the requirements of access control and protection in context-aware computing environments, we presented the Activity-based Access Control model to provide security designers a conceptual view of access control design requirements. Abstracting access control elements allows designers to prove their system is secure. We also expressed how the model helps express complex access control policies through definitions of context constraints and activity constraints. The benefits of ABAC over many other current models are its simplicity. ABAC is simple because the notion of an activity is intuitive and designing and encapsulating constraints around the concept is not difficult to create. ABAC hides implementation details away from the designer, and requires minimum interaction from users above what RBAC requires. The ABAC model also allows for both coarse-grained and fine-grained access control policies. At a minimum, it fulfills RBAC functionality. If an organization wishes to utilize context constraints and prevent collisions of current user activities, a designer may include additional constraints provided by the ABAC model.

5.2 Thesis Contributions

In summary, the contributions of this thesis are:

- A framework for Pervasive Computing environment with definitions of context-awareness and PC components. The framework was formulated through our ideas and by multitudes of research in the area.

- Introduction of a problem present in Pervasive Computing. That is, we presented the consistency problem involving the parallel usage of services which may cause conflicts in any Pervasive Computing environment.
- Introduction to the Activity-based Access Control model, used to design context-aware policies for users and applications in PC environments.
- An implementation and validation of the ABAC model enforcing role cardinality, activity mutual exclusion, activity context, and role context constraints.

5.3 Future Direction

There is still much work before a Pervasive Computing environment is deployed. Our concentration was towards a context-aware access control system for the environment, but to achieve accurate understanding and validation of the system, a Pervasive Computing infrastructure is required. Future directions of this research are the context constraint constructions and validation to allow for efficiency, consistency, security, and timeliness. Additionally, an assumption made in this thesis was the existence of the authentication service. In order to confirm the system, design, development, and integration of brokers, context providers, PC environment services, user applications, and resources is needed.

We studied the challenges involved with development of a universal Pervasive Computing infrastructure and providing security services to entities in Pervasive Computing environments. We have presented our view of a Pervasive Computing environment and a context-aware access control model for security designers. Our model provides various constraints allowing for fine-grained policies when needed. Additionally, we have provided an ABAC implementation to demonstrate the effectiveness of our model.

APPENDIX

Appendix A

Consultant Phone Meeting Script

```
#######################

# ENTITIES

#######################

# generate some users

ADD USER bob

ADD USER susan


#generate some roles

ADD ROLE consultant

ADD ROLE client


# generate some activities

ADD ACTIVITY private_phone_meeting


# generate some constraints

ADD CONSTRAINT persons_in_room


# generate some conditions

ADD CONDITION single_person

              ((context('number_people', 'room_320') = '1')

ADD CONDITION consultant_location all('role', 'consultant',
              ((context('location', 'consultant') = 'room_320'))


# generate some context

ADD CONTEXT number_people

ADD CONTEXT location


# generate some subject type

ADD SUBJECTTYPE room

ADD SUBJECTTYPE user

ADD SUBJECTTYPE role


# generate some subject

ADD SUBJECT room_320 room

ADD SUBJECT consultant role

ADD SUBJECT client role

ADD SUBJECT bob user

ADD SUBJECT susan user


#######################

# RELATIONSHIPS

#######################
```

```
# assign users to roles

ASSIGN USER susan client

ASSIGN USER bob consultant


# add activity roles

ADD ACTIVITYROLE private_phone_meeting client 1 1

ADD ACTIVITYROLE private_phone_meeting consultant 1 1


# associate conditions to constraints

ADD CONSTRAINTCONDITION persons_in_room single_person

ADD CONSTRAINTCONDITION persons_in_room consultant_location


# associate conditions to context_subjects

ADD CONDITIONCONTEXTSUBJECT single_person number_people

                                          room_320 room

ADD CONDITIONCONTEXTSUBJECT consultant_location location

                                          consultant role

ADD CONDITIONCONTEXTSUBJECT consultant_location location

                                          client role


# associate constraint to activity
```

```
ADD ACTIVITYCONSTRAINT private_phone_meeting persons_in_room


#########################

# ACTIVATIONS

#########################

# create some sessions

ADD SESSION susan consultant_chat

ADD SESSION bob client_chat


# activate session roles

ACTIVATE susan consultant_chat client

ACTIVATE bob client_chat consultant


# associate sessions to activity: join activity request

ADD SESSIONACTIVITY private_phone_meeting consultant_chat

                                          susan

ADD SESSIONACTIVITY private_phone_meeting  client_chat bob


# activate context receive

ACTIVATE CONTEXT number_people room_320 room_320 room

ACTIVATE CONTEXT location susan client role
ACTIVATE CONTEXT location bob consultant role


# send initial context to access control system

UPDATE CONTEXT number_people room_320 1

UPDATE CONTEXT location susan room_125

UPDATE CONTEXT location bob room_325


###############################

# DEACTIVATIONS & DELETIONS

###############################

# Unsubscribe from context

DEACTIVATE CONTEXT number_people room_320

DEACTIVATE CONTEXT location susan

DEACTIVATE CONTEXT location bob


# Resign sessions from activity

DELETE SESSIONACTIVITY  private_phone_meeting consultant_chat susan

DELETE SESSIONACTIVITY  private_phone_meeting client_chat bob


# Resign session roles

DEACTIVATE susan consultant_chat client
DEACTIVATE bob client_chat consultant


# delete sessions

DELETE SESSION susan consultant_chat

DELETE SESSION bob client_chat


# deassign users from roles

DEASSIGN USER susan client

DEASSIGN USER bob consultant


# Unsassign activity roles

DELETE ACTIVITYROLE private_phone_meeting client

DELETE ACTIVITYROLE private_phone_meeting consultant


# Unassign activity constraint

DELETE ACTIVITYCONSTRAINT private_phone_meeting persons_in_room


# delete activities

DELETE ACTIVITY private_phone_meeting


# delete roles
```

```
DELETE ROLE consultant

DELETE ROLE client


# delete users

DELETE USER bob

DELETE USER susan


# delete condition and context-subject association

DELETE CONDITIONCONTEXTSUBJECT single_person number_people

                                             room_320 room

DELETE CONDITIONCONTEXTSUBJECT consultant_location location

                                             consultant role

DELETE CONDITIONCONTEXTSUBJECT consultant_location location

                                             client role


# delete constraint-conditions

DELETE CONSTRAINTCONDITION persons_in_room single_person

DELETE CONSTRAINTCONDITION persons_in_room consultant_location


# delete constraint

DELETE CONSTRAINT persons_in_room
```

```
# delete conditions

DELETE CONDITION single_person

DELETE CONDITION consultant_location


# delete contexts

DELETE CONTEXT number_people

DELETE CONTEXT location


# delete context subjects

DELETE SUBJECT room_320 room

DELETE SUBJECT room_320 consultant

DELETE SUBJECT room_320 client

DELETE SUBJECT bob user

DELETE SUBJECT susan user


# delete subject types

DELETE SUBJECTTYPE room

DELETE SUBJECTTYPE role

DELETE SUBJECTTYPE user
```

```
QUIT
```

Appendix B

Rated Movie Policy & Script

```
#######################

# ENTITIES

#######################

# generate some users

ADD USER jack

ADD USER jill

ADD USER mary


#generate some roles

ADD ROLE adult

ADD ROLE child


# generate some objects

ADD OBJECT movie


# generate some operations

ADD OPERATION watch


# generate permission

ADD PERMISSION movie watch
```

```
# generate some activities

ADD ACTIVITY rated_r_television


# generate some constraints

ADD CONSTRAINT parent_advisory


# generate some conditions

ADD CONDITION adult_location exist('role', 'adult',

((context('location', 'adult') = 'living_room'))


# generate some context

ADD CONTEXT location


# generate some subject type

ADD SUBJECTTYPE role

ADD SUBJECTTYPE user

ADD SUBJECTTYPE room


# generate some subject

ADD SUBJECT living_room room

ADD SUBJECT child role
ADD SUBJECT adult role

ADD SUBJECT jill user

ADD SUBJECT jack user

ADD SUBJECT mary user


#######################

# RELATIONSHIPS

#######################

# assign users to roles

ASSIGN USER mary child

ASSIGN USER jack adult

ASSIGN USER jill adult


# add activity roles

ADD ACTIVITYROLE rated_r_television adult 1 256

ADD ACTIVITYROLE rated_r_television child 0 256


# associate conditions to constraints

ADD CONSTRAINTCONDITION parent_advisory adult_location


# associate conditions to context_subjects
```

```
ADD CONDITIONCONTEXTSUBJECT adult_location location adult role


# associate constraint to activity

ADD ACTIVITYCONSTRAINT rated_r_television parent_advisory


# grant permission

GRANT adult movie watch

GRANT child movie watch


#########################

# ACTIVATIONS

#########################

# create some sessions

ADD SESSION mary child_tv

ADD SESSION jack adult_tv_1

ADD SESSION jill adult_tv_2


# activate session roles

ACTIVATE jack adult_tv_1 adult

ACTIVATE jill adult_tv_2 adult

ACTIVATE mary child_tv child
```

```
# associate sessions to activity: join activity request

ADD SESSIONACTIVITY rated_r_television adult_tv_1 jack

ADD SESSIONACTIVITY rated_r_television adult_tv_2 jill

ADD SESSIONACTIVITY rated_r_television child_tv mary


# activate context receive

ACTIVATE CONTEXT location jack adult role

ACTIVATE CONTEXT location jill adult role

ACTIVATE CONTEXT location mary child role


################################

# DEACTIVATIONS & DELETIONS

################################

# Unsubscribe from context

DEACTIVATE CONTEXT location jack

DEACTIVATE CONTEXT location jill

DEACTIVATE CONTEXT location mary


# Resign sessions from activity

DELETE SESSIONACTIVITY rated_r_television adult jack
DELETE SESSIONACTIVITY rated_r_television adult jill

DELETE SESSIONACTIVITY rated_r_television child mary


# Resign session roles

DEACTIVATE jack adult_tv_1 adult

DEACTIVATE jill adult_tv_2 adult

DEACTIVATE mary child_tv child


# delete sessions

DELETE SESSION mary child_tv

DELETE SESSION jack adult_tv_1

DELETE SESSION jill adult_tv_2


# revoke permission

REVOKE adult movie watch

REVOKE child movie watch


# Unassign activity constraint

DELETE ACTIVITYCONSTRAINT rated_r_television adult_location


# deassign users from roles
```

```
DEASSIGN USER mary child

DEASSIGN USER jack adult

DEASSIGN USER jill adult


# Unsassign activity roles

DELETE ACTIVITYROLE adult_tv_1 adult

DELETE ACTIVITYROLE adult_tv_2 adult

DELETE ACTIVITYROLE child_tv child


# delete condition and context-subject association

DELETE CONDITIONCONTEXTSUBJECT adult_location location adult role


# delete constraint-conditions

DELETE CONSTRAINTCONDITION parent_advisory adult_location


# delete contexts

DELETE CONTEXT location


# delete context subjects

DELETE SUBJECT living_room room

DELETE SUBJECT child role
DELETE SUBJECT adult role

DELETE SUBJECT jill user

DELETE SUBJECT jack user

DELETE SUBJECT mary user


# delete subject types

DELETE SUBJECTTYPE role

DELETE SUBJECTTYPE user

DELETE SUBJECTTYPE room


# delete conditions

DELETE CONDITION adult_location


# delete constraints

DELETE CONSTRAINT parent_advisory


# delete activities

DELETE ACTIVITY rated_r_television


# delete permission

DELETE PERMISSION movie watch
```

```
# delete operations

DELETE OPERATION watch


# delete objects

DELETE OBJECT movie


# delete roles

DELETE ROLE adult

DELETE ROLE child


# delete users

DELETE USER jack

DELETE USER jill

DELETE USER mary


QUIT
```

BIBLIOGRAPHY

[1] J. Al-Muhtadi, A. Ranganathan, R. H. Campbell, and M. D. Mickunas. Cerberus: A context-aware security scheme for smart spaces. In *PerCom*, pages 489–, 2003.

[2] F. Alotaiby and J. Chen. A model for team-based access control (TMAC 2004). In *Information Technology: Coding and Computing, 2004. Proceedings. ITCC 2004.*, volume 1, pages 450–454, 2004.

[3] J. Bacon, K. Moody, and W. Yao. A model of oasis role-based access control and its support for active security. *ACM Trans. Inf. Syst. Secur.*, 5(4):492–540, 2002.

[4] G. Banavar, J. Beck, E. Gluzberg, J. Munson, J. Sussman, and D. Zukowski. Challenges: an application model for pervasive computing. In *MobiCom '00: Proceedings of the 6th annual international conference on Mobile computing and networking*, pages 266–274, New York, NY, USA, 2000. ACM Press.

[5] J. Bardram, R. E. Kjær, and M. Ø. Pedersen. Context-aware user authentication - supporting proximity-based login in pervasive computing. In *Ubicomp*, pages 107–123, 2003.

[6] J. Barkley. Comparing simple role based access control models and access control lists. In *RBAC '97: Proceedings of the second ACM workshop on Role-based access control*, pages 127–132, New York, NY, USA, 1997. ACM Press.

[7] C. Braz. Authenlink: a user-centred authentification system for a secure mobile com. In *Proceedings of the 1st French-speaking conference on Mobility and ubiquity computing*, pages 17–20, New York, NY, USA, 2004. ACM Press.

[8] P. J. Brown, J. D. Bovey, and X. Chen. Context-aware applications: from the laboratory to the marketplace. *IEEE Personal Communications*, 4(5):58–64, October 1997.

[9] A. Bullock and S. Benford. An access control framework for multi-user collaborative environments. In *GROUP '99: Proceedings of the international ACM SIGGROUP conference on Supporting group work*, pages 140–149, New York, NY, USA, 1999. ACM Press.

[10] R. H. Campbell, J. Al-Muhtadi, P. Naldurg, G. Sampemane, and M. D. Mickunas. Towards security and privacy for pervasive computing. In *ISSS*, pages 1–15, 2002.

[11] P. Castro, B. Greenstein, R. Muntz, P. Kermani, C. Bisdikian, and M. Papadopouli. Locating application data across service discovery domains. In *MobiCom '01: Proceedings of the 7th annual international conference on Mobile computing and networking*, pages 28–42, New York, NY, USA, 2001. ACM Press.

[12] E. Cohen, R. K. Thomas, W. Winsborough, and D. Shands. Models for coalition-based access control (cbac). In *SACMAT '02: Proceedings of the seventh ACM symposium on Access control models and technologies*, pages 97–106, New York, NY, USA, 2002. ACM Press.

[13] M. J. Covington, P. Fogla, Z. Zhan, and M. Ahamad. A context-aware security architecture for emerging applications. In *ACSAC*, pages 249–260, 2002.

[14] M. J. Covington, W. Long, S. Srinivasan, A. K. Dev, M. Ahamad, and G. D. Abowd. Securing context-aware applications using environment roles. In *SACMAT '01: Proceedings of the sixth ACM symposium on Access control models and technologies*, pages 10–20, New York, NY, USA, 2001. ACM Press.

[15] O. Davidyuk, J. Riekki, V.-M. Rautio, and J. Sun. Context-aware middleware for mobile multimedia applications. In *MUM '04: Proceedings of the 3rd international conference on Mobile and ubiquitous multimedia*, pages 213–220, New York, NY, USA, 2004. ACM Press.

[16] Department of Defense. *Department of Defense Trusted Computer System Evaluation Criteria*, Dec. 1985. DOD 5200.28-STD (supersedes CSC-STD-001-83).

[17] A. Dey. Context-aware computing: The cyberdesk project. In *Proceedings of AAAI '98 Spring Symposium*, pages 51–54, March 1998.

[18] A. K. Dey. Understanding and using context. *Personal Ubiquitous Computing*, 5(1):4–7, 2001.

[19] A. K. Dey and J. Mankoff. Designing mediation for context-aware applications. *ACM Transaction Computer-Humam Interaction*, 12(1):53–80, 2005.

[20] D. F. Ferraiolo, J. A. Cugini, and D. R. Kuhn. Role-based access control (RBAC): Features and motivations. In *11 th Annual Computer Security Applications Proceedings*, 1995.

[21] D. F. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramouli. Proposed nist standard for role-based access control. *ACM Trans. Inf. Syst. Secur.*, 4(3):224–274, 2001.

[22] S. Fickas, G. Kortuem, and Z. Segall. Software organization for dynamic and adaptable wearable systems. In *ISWC '97: Proceedings of the 1st IEEE International Symposium on Wearable Computers*, page 56, Washington, DC, USA, 1997. IEEE Computer Society.

[23] A. Friday, N. Davies, N. Wallbank, E. Catterall, and S. Pink. Supporting service discovery, querying and interaction in ubiquitous computing environments. *Wirel. Netw.*, 10(6):631–641, 2004.

[24] C. K. Georgiadis, I. Mavridis, G. Pangalos, and R. K. Thomas. Flexible team-based access control using contexts. In *SACMAT '01: Proceedings of the sixth ACM symposium on Access control models and technologies*, pages 21–27, New York, NY, USA, 2001. ACM Press.

[25] G.Kortuem, Z. Segall, and M. Bauer. Context-aware, adaptive wearable computers as remote interfaces to 'intelligent' environments. In *ISWC*, pages 58–65, 1998.

[26] J. M. Haake, A. Haake, T. Schümmer, M. Bourimi, and B. Landgraf. End-user controlled group formation and access rights management in a shared workspace system. In *CSCW '04: Proceedings of the 2004 ACM conference on Computer supported cooperative work*, pages 554–563, New York, NY, USA, 2004. ACM Press.

[27] J. I. Hong. The context fabric: an infrastructure for context-aware computing. In *CHI '02: CHI '02 extended abstracts on Human factors in computing systems*, pages 554–555, New York, NY, USA, 2002. ACM Press.

[28] M. C. Huebscher and J. A. McCann. Adaptive middleware for context-aware applications in smart-homes. In *Proceedings of the 2nd workshop on Middleware for pervasive and ad-hoc computing*, pages 111–116, New York, NY, USA, 2004. ACM Press.

[29] R. Hull, P. Neaves, and J. Bedford-Roberts. Towards situated computing. In *ISWC '97: Proceedings of the 1st IEEE International Symposium on Wearable Computers*, page 146, Washington, DC, USA, 1997. IEEE Computer Society.

[30] S. Khungar and J. Riekki. A context based storage for ubiquitous computing applications. In *EUSAI '04: Proceedings of the 2nd European Union symposium on Ambient intelligence*, pages 55–58, New York, NY, USA, 2004. ACM Press.

[31] D. R. Kuhn. Mutual exclusion of roles as a means of implementing separation of duty in

role-based access control systems. In *RBAC ’97: Proceedings of the second ACM workshop on Role-based access control*, pages 23–30, New York, NY, USA, 1997. ACM Press.

[32] B. Lampson. Protection. In *Proceedings of the 5th Annual Princeton Conference on Information Sciences and Systems*, pages 437–443, Princeton University, 1971.

[33] B. W. Lampson. A note on the confinement problem. *Commun. ACM*, 16(10):613–615, 1973.

[34] C. E. Landwehr. Formal models for computer security. *ACM Comput. Surv.*, 13(3):247–278, 1981.

[35] H. Lei, D. M. Sow, I. John S. Davis, G. Banavar, and M. R. Ebling. The design and applications of a context service. *SIGMOBILE Mob. Comput. Commun. Rev.*, 6(4):45–55, 2002.

[36] G. Mostefaoui, J. Pasquier-Rocha, and P. Brezillon. Context-aware computing: a guide for the pervasive computing community. In *The IEEE/ACS International Conference on Pervasive Services (ICPS 2004)*, pages 39 – 48, 2004.

[37] G. Neumann and M. Strembeck. An approach to engineer and enforce context constraints in an RBAC environment. In *SACMAT ’03: Proceedings of the eighth ACM symposium on Access control models and technologies*, pages 65–79, New York, NY, USA, 2003. ACM Press.

[38] L. Nigay and P. Gray. Architecture logicielle conceptuelle pour la capture de contexte. In *IHM ’02: Proceedings of the 14th French-speaking conference on Human-computer interaction (Confrence Francophone sur l’Interaction Homme-Machine)*, pages 211–214, New York, NY, USA, 2002. ACM Press.

[39] S. Oh and S. Park. Task-role-based access control model. *Inf. Syst.*, 28(6):533–562, 2003.

[40] S. Osborn. Mandatory access control and role-based access control revisited. In *RBAC '97: Proceedings of the second ACM workshop on Role-based access control*, pages 31–40, New York, NY, USA, 1997. ACM Press.

[41] J. S. Park and J. Hwang. Role-based access control for collaborative enterprise in peer-to-peer computing environments. In *SACMAT '03: Proceedings of the eighth ACM symposium on Access control models and technologies*, pages 93–99, New York, NY, USA, 2003. ACM Press.

[42] J. Pascoe. Adding generic contextual capabilities to wearable computers. In *The Second International Symposium on Wearable Computers*, pages 92–99, Pittsburgh, October 1998. IEEE Computer Society. Online proceedings available from http://iswc.gatech.edu/.

[43] A. Ranganathan, R. Campbell, A. Ravi, and A. Mahajan. Conchat: A context-aware chat program. *IEEE Pervasive Computing*, pages 52–58, July-Sept 2002.

[44] A. Ranganathan and R. H. Campbell. An infrastructure for context-awareness based on first order logic. *Personal Ubiquitous Comput.*, 7(6):353–364, 2003.

[45] G. Rey and J. Coutaz. Le contexteur: une abstraction logicielle pour la réalisation de systèmes interactifs sensibles au contexte. In *IHM '02: Proceedings of the 14th French-speaking conference on Human-computer interaction (Conférence Francophone sur l'Interaction Homme-Machine)*, pages 105–112, New York, NY, USA, 2002. ACM Press.

[46] J. Robinson, I. Wakeman, and T. Owen. Scooby: middleware for service composition in pervasive computing. In *Proceedings of the 2nd workshop on Middleware for pervasive and ad-hoc computing*, pages 161–166, New York, NY, USA, 2004. ACM Press.

[47] N. S. Ryan, J. Pascoe, and D. R. Morse. Enhanced reality fieldwork: the context-aware archaeological assistant. In V. Gaffney, M. van Leusen, and S. Exxon, editors, *Computer*

Applications in Archaeology 1997, British Archaeological Reports, Oxford, October 1998. Tempus Reparatum.

[48] D. Salber, A. K. Dey, and G. D. Abowd. The context toolkit: aiding the development of context-enabled applications. In *CHI '99: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 434–441, New York, NY, USA, 1999. ACM Press.

[49] D. Salber, A. K. Dey, and G. D. Abowd. The context toolkit: aiding the development of context-enabled applications. In *CHI '99: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 434–441, New York, NY, USA, 1999. ACM Press.

[50] G. Sampemane, P. Naldurg, and R. H. Campbell. Access control for active spaces. In *ACSAC*, pages 343–352, 2002.

[51] R. Sandhu and Q. Munawer. How to do discretionary access control using roles. In *RBAC '98: Proceedings of the third ACM workshop on Role-based access control*, pages 47–54, New York, NY, USA, 1998. ACM Press.

[52] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, Feb. 1996.

[53] M. Satyanarayanan. Pervasive computing; vision and challenges. *ieee-pcm*, 8(4):10–17, August 2001.

[54] B. Schilit, N. Adams, and R. Want. Context-aware computing applications. *First International Workshop on Mobile Computing Systems and Applications*, pages 85–90, 1994.

[55] B. Schilit and M. Theimer. Disseminating active map information to mobile hosts. *IEEE Network*, 8(5):22–32, 1994.

[56] C. Schlenoff and M. Uschold, editors. *A Context Broker for Building Smart Meeting Rooms*, Stanford, California, March 2004. AAAI, AAAI Press, Menlo Park, CA.

[57] H. Shen and P. Dewan. Access control for collaborative environments. In *CSCW '92: Proceedings of the 1992 ACM conference on Computer-supported cooperative work*, pages 51–58, New York, NY, USA, 1992. ACM Press.

[58] Z. Song, R. Masuoka, J. Agre, and Y. Labrou. Task computing for ubiquitous multimedia services. In *MUM '04: Proceedings of the 3rd international conference on Mobile and ubiquitous multimedia*, pages 257–262, New York, NY, USA, 2004. ACM Press.

[59] M. Strembeck and G. Neumann. An integrated approach to engineer and enforce context constraints in RBAC environments. *ACM Trans. Inf. Syst. Secur.*, 7(3):392–427, 2004.

[60] R. K. Thomas. Team-based access control (TMAC): a primitive for applying role-based access controls in collaborative environments. In *RBAC '97: Proceedings of the second ACM workshop on Role-based access control*, pages 13–19, New York, NY, USA, 1997. ACM Press.

[61] R. K. Thomas and R. S. Sandhu. Conceptual foundations for a model of task-based authorizations. In *Proceedings of the 7th IEEE Computer Security Foundations Workshop*, pages 66–79, 1994.

[62] R. K. Thomas and R. S. Sandhu. Task-based authorization controls (tbac): A family of models for active and enterprise-oriented autorization management. In *Proceedings of the IFIP TC11 WG11.3 Eleventh International Conference on Database Securty XI*, pages 166–181, London, UK, UK, 1997. Chapman & Hall, Ltd.

[63] W. Tolone, G.-J. Ahn, T. Pai, and S.-P. Hong. Access control in collaborative systems. *ACM Comput. Surv.*, 37(1):29–41, 2005.

[64] A. Tripathi, T. Ahmed, D. Kulkarni, R. Kumar, and K. Kashiramka. Context-based secure resource access in pervasive computing environments. In *Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications Workshops*, page 159. IEEE Computer Society, 2004.

[65] W. Wang. Team-and-role-based organizational context and access control for cooperative hypermedia environments. In *HYPERTEXT '99: Proceedings of the tenth ACM Conference on Hypertext and hypermedia : returning to our diverse roots*, pages 37–46, New York, NY, USA, 1999. ACM Press.

[66] R. Want, A. Hopper, V. Falcão, and J. Gibbons. The active badge location system. Technical Report 92.1, Olivetti Research Ltd. (ORL), 24a Trumpington Street, Cambridge CB2 1QA, 1992.

[67] A. Ward, A. Jones, and A. Hopper. A new location technique for the active office, 1997.

[68] H. F. Wedde and M. Lischka. Role-based access control in ambient and remote space. In *SACMAT*, pages 21–30, 2004.

[69] M. Weiser. The computer for the 21st century. pages 933–940, 1995.

[70] G. Zhang and M. Parashar. Dynamic context-aware access control for grid applications. In *Proceedings of the Fourth International Workshop on Grid Computing*, page 101. IEEE Computer Society, 2003.

[71] 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)*, 2004.