Context-Aware Role-based Access Control in Pervasive Computing Systems

Devdatta Kulkarni and Anand Tripathi[*]
Dept. of Computer Science, University of Minnesota
Twin Cities, MN 55455, USA
(dkulk,tripathi)@cs.umn.edu

ABSTRACT

In this paper we present a context-aware RBAC (CA-RBAC) model for pervasive computing applications. The design of this model has been guided by the context-based access control requirements of such applications. These requirements are related to users' memberships in roles, permission executions by role members, and context-based dynamic integration of services in the environment with an application. Context information is used in role admission policies, in policies related to permission executions by role members, and in policies related to accessing of dynamically interfaced services by role members. The dynamic nature of context information requires model-level support for revocations of role memberships and permission activations when certain context conditions fail to hold. Based on this model we present a programming framework for building context-aware applications, providing mechanisms for specifying and enforcing context-based access control requirements.

Categories and Subject Descriptors

K.6.5 [**Management of Computing and Information Systems**]: Security and Protection; D.2.6 [**Software Engineering**]: Programming Environments

General Terms

Design, Experimentation, Languages, Security

Keywords

Context-Aware Computing, Pervasive Computing, Context-based Access Control, RBAC

1. INTRODUCTION

Context-awareness is a central aspect of pervasive computing applications, characterizing their ability to adapt and perform tasks based on ambient context conditions. There has been steady adoption of context-awareness in number of application domains such as assisted living [8], hospital information systems [5], tour guides [11], and smart environments [30]. At the same time, there has been an increasing concern among researchers for security requirements of such applications [7]. Especially, in the domain of medical information systems such concerns have been recognized for a long time [32, 25], and system models have been developed for access control in such applications [4, 16].

Various definitions of *context* have been proposed in the literature [1, 29]. Broadly, the notion of *context* in pervasive computing applications relates to the characterization of ambient conditions and physical world situations that are relevant for performing appropriate actions in the computing domain for its correct or desired behavior. A person's context is defined in terms of his/her current physical location, devices being used, network on which the devices are connected, and the activities in which the user is currently engaged. Additionally, there can be other conditions and characteristics that may be relevant in defining a context. For example, in some situations the temporal attributes associated with an activity, such as its duration and time of occurrence, may be important. Other factors such as device capabilities, physical proximity of devices, and available bandwidth can also be important in some situations. Some of the typical context-based adaptive characteristics include [29] dynamic integration of resources/services with an application based on context information, context-based access control, displaying information based on the context information, and context-triggered actions.

The focus of this paper is on the issues related to building role-based access control models and systems for pervasive computing applications, utilizing context information in making access control decisions. Several researchers have developed RBAC models that support context-based access control [13, 24, 28, 6, 9, 4, 16, 21]. Context-based constraints may be specified on the User-Assignment (UA) relation [21], or on the activation of role sessions [13], or on the Permission-Assignment (PA) relation [24]. Different kinds of context information have been considered as part of building RBAC systems. These include users' presence in an active space [28], users' presence in specific geographic areas [6], occurrence of specified environmental conditions [9],

[*] This work was supported by NSF grants 0411961, 0708604.

SACMAT'08, June 11–13, 2008, Estes Park, Colorado, USA.
Copyright 2008 ACM 978-1-60558-129-3/08/06 ...$5.00.

a user's memberships in other roles [4] or teams [16], or time intervals as in the GTRBAC model [21].

Integrating context information as part of an access control system is a challenging task due to several reasons. First, acquiring appropriate context information requires interfacing the access control system with various kinds of ambient sensors. Integrity and authenticity of this information is paramount because it may be used in making access control decisions. Second, certain aspects of the context information may be inherently dynamic in nature. During the course of execution of a context-dependent task, it is possible for the related context condition to become false. For certain applications, it may be important that a role member's permissions to execute that task are revoked when such context changes occur. Third, context-based constraints may restrict the resources and services that may be dynamically interfaced with a pervasive computing application.

Pervasive computing applications considered here are deployed in an *active space* containing various kinds of devices, services, and ambient sensors. Context management services are deployed to aggregate sensor data for detecting application-defined context conditions. Resource discovery services are also deployed in an active space. Other active space services register with the discovery services. Applications use the discovery services to dynamically discover and interface with appropriate services in a given context situation. We refer to this as *dynamic object binding*. Applications may define context-based access control policies for such dynamically interfaced services.

The main contribution of this paper is a context-aware RBAC (CA-RBAC) model and its embodiment in a programming framework for designing context-aware applications. The novel features of this programming framework are high-level abstractions for specifying context-based access control requirements, specifically addressing the following aspects:

1. *Role admission and validation constraints:* These constraints specify context-based conditions that need to be satisfied before admitting a user to a role, and also for continuing a user's membership in a role.

2. *Context-based role permissions:* Dynamic object binding causes role operations to interface with different services under different context conditions.

3. *Personalized role permissions:* Such permissions allow different role members to access different active space services based on their individual context.

4. *Context-based permission activation constraints:* These constraints are associated with specific role permissions, and specify context-based conditions that need to hold for a role member to execute such permissions.

5. *Context-based resource access constraints:* These constraints restrict a role member's access to a subset of resources that are managed by an active space service.

The paper is organized as follows. In Section 2 we present two representative context-aware applications which we use to demonstrate the above kinds of access control requirements. In Section 3 we present the CA-RBAC model. In Section 4 we present our programming framework that realizes this model. We compare our work with the related work in Section 5, and conclude in Section 6.

2. RBAC REQUIREMENTS FOR CONTEXT-AWARE COMPUTING

Here we present two representative context-aware applications to motivate the need for extending the NIST RBAC model [13] to support context-based access control requirements.

Context-Aware Patient Information System: Consider a patient information system deployed in a hospital and accessed by the hospital nurses and doctors. It supports a number of different requirements as follows [12]. The system supports assigning a nurse to a ward during certain time periods. During these periods the assigned nurse works in the capacity of a nurse-on-duty role in that ward. In this role the system permits access to the records of only those patients who are admitted to the ward where the nurse is currently present. The nurse's membership in the nurse-on-duty role is revoked when she leaves the ward, or after the end of her duty time. The system permits doctors to create different kinds of reports about patients. For a nurse, access to doctor's reports is allowed only if some doctor is present in the ward where the nurse is located. It may happen that a nurse initiates access to the doctors' reports while a doctor is present in the ward, but the doctor leaves the ward afterwards. In such a case the nurse's on-going session accessing such reports needs to be terminated. This ensures that a nurse does not continue to access these reports in the absence of a doctor. This system may also support location-based access to active space services by nurses and doctors.

Context-Aware Music Player: Consider a music player application which runs on a user's mobile device. Depending on the user's physical location, it streams music either to the user's device or to the audio player service in the room where the user is currently present. We consider the following context-based access control requirements for this application. When the user enters a room, the access control system should allow the application to automatically start streaming music to that room's audio player service if no other user is present in the room. The access control system must revoke the application's access to the room's audio player service when the user leaves the room or some other person enters the room.

2.1 Role Model

Role-based models such as the NIST RBAC have been traditionally used for designing access control systems for organizations. In such systems, roles generally have a long lifetime. Users are assigned to a role by the system administrator, and such memberships also tend to have long duration. In contrast to this, in our model roles are defined as part of an application's design. Such roles come into existence only when that application is deployed and executed, and they last only during the application's lifetime. In our model, we use the following RBAC terminology [2]. Users may be *added* to the roles at the time of application deployment, or they may request to *join* a role when the application is executing. The access control system underlying the application execution environment *admits* a user to a role based on the role admission constraints. A user admitted to a role is called a *role member*. The permissions associated with a role are represented by a set of operations through which a role member may access an *object*. The execution of such operations by a role member amounts to role *activation*. In this model, there is no explicit notion of *sessions* as in the NIST model. This is because the static and dynamic separation-of-duty constraints can be specified using *event history* based constraints on role operation executions [2]. In some applications, a user's membership in a role may be transient, as it may need to be revoked depending on the context conditions.

2.2 Role admission and validation

User admission to a role may be based on different kinds of context information such as temporal constraints, prior membership in other roles, or user location. Examples of each of these constraints are seen in the patient information system. In this application we may define different roles such as *Nurse* and *Doctor*. We may also define a *NurseOnDuty* role for different wards. Temporal constraints may be used to restrict user membership in the *NurseOnDuty* role only during certain time periods. Prior role membership constraints may be used to restrict only the members of the *Nurse* role to be admitted to the *NurseOnDuty* role for a particular ward. A role member's physical location may be used to allow only those nurses who are physically present in a ward to be admitted to the *NurseOnDuty* role for that ward.

A complementary aspect to context-based role admission is the need to revoke a user's role memberships when specified context conditions fail to hold. For example, in the patient information system a nurse's membership in the *NurseOnDuty* role needs to be revoked when the nurse leaves that ward or after expiration of the specified time interval. We call the above requirement as *role validation constraint*. It determines the validity of a user's membership in a role.

In the NIST RBAC model, there is no explicit notion of role membership revocations. Role membership revocation requirements have been considered by the OASIS RBAC model [4], where a user's membership in a role may be contingent on the membership in some other roles. Some models [21, 6] have used the notion of deactivating a role under certain context conditions. A deactivated role becomes inaccessible to all of its members. In our model, a specific user's privileges for a role are revoked by removing the user from the role membership. Such role membership revocation can be crucial for enforcing policies that may be sensitive to role membership cardinalities.

2.3 Context-based role permissions

In the NIST RBAC model the set of objects for which access control needs to be enforced are statically known. However, in pervasive computing applications specific services that may be accessed by a role member may not be known a priori. They may depend on the context information associated with an application. The application may need to *discover* an appropriate service in the active space and grant access to it under certain context conditions. Because the specific services with which the application would be interfaced is generally not known a priori, the permissions may only be specified on an abstract object. At runtime, this object is dynamically bound to a specific service available in the active space.

We see above kind of requirement in the context-aware music player application. We need the music to stream either to the user's device or to the audio player service in the room where the user is present. Moreover, we need the music to stream to the room's audio player service only if no other person is present in the room. To program this requirement we may define an abstract object named *audio player* to represent the service through which the music would be played. This object would be dynamically bound either to the audio player service in the room where the user is present, or to the audio player service on the user's device. We may also define a *User* role and provide the permission to *play* music on the *audio player* object. As part of binding the object we first authenticate the audio player service and then check for its compatibility to allow invocation of the permission to play music.

2.4 Personalized role permissions

The services that are accessible through a role permission may be different for different role members and may depend on the context information associated with a role member. In the NIST RBAC model the set of objects accessed through a permission are always the same for all the members of a role. A role permission invoked by any member of a role is executed on the *same* object. However, this model is inadequate for pervasive computing applications where a permission invoked by *different* role members may need to be invoked on *different* object instances based on each role member's individual context, such as the physical location.

As an example consider the permission to *print* which is associated with the *Nurse* role in the patient information system. We may require that when this permission is invoked by a nurse, the system chooses the most appropriate printer for satisfying that request. The choice of a particular printer for a specific nurse may depend on various things such as the nurse's preference for a specific printer, context information such as the nurse's location, or the physical security of the printer, or the security level associated with the material being printed.

2.5 Context-based permission activation

Execution of role operations by role members may need to be constrained based on context conditions. The access control model needs to support specification of context-based constraints that need to be satisfied *before* a role member may execute an operation.

Certain operation executions may lead to interactive sessions of arbitrary duration with one or more objects. In some applications we may need such sessions to remain active only under certain context conditions. Correspondingly, the access control model needs to provide mechanisms to terminate sessions initiated through the operation execution when the corresponding context conditions fail to hold. We refer to such changes in required context conditions as *context invalidations*.

We see both the above requirements in the patient information system and in the music player application. In the patient information system we need to restrict a nurse's access to patient reports only if the nurse is co-located with a doctor in a hospital ward. This requirement can be modeled as a context-based constraint on operation execution by nurses. The invocation of the operation by a nurse to access patient reports may lead to initiation of a session with the *database service*. We may require that this session should be terminated when either the nurse or the doctor leaves the ward. This is an example of context invalidation.

In the music player application we require that the access control system should grant access for a room's audio player service to a user only if that user and no other person is present in that room. This can be modeled as a context-based constraint on permission to play the music by the application role. The context condition corresponding to the user being alone in a room is invalidated when some other person enters the room while the music is being played on the room's audio player service. In this case the access control system needs to revoke the application's access to the room's audio player service. Moreover, this access also needs to be revoked when the user leaves the room.

We observe that currently such context-based permission activations and revocations are not supported in the NIST RBAC model. The following mechanisms are needed for this purpose. First, we need mechanisms to integrate context information as part of constraint specification on permission activation in a RBAC model. Second, we need mechanisms to continuously monitor context conditions that need to hold while a permission session is active. Third, we need revocation mechanisms to terminate such sessions when the corresponding context conditions fail to hold.

2.6 Context-based resource access

There is a need to distinguish between a service and a resource for access control purpose. A service may be managing a number of resources of a specific type. We may need to control a role member's access to a subset of these resources based on context conditions. For example, in the patient information system the *database service* controls access to the database tables and determines who gets access to them and under what conditions. Such a database may store information about patients such as doctor reports, prescriptions, tests performed, last checkup time, and patient's ward. In this application we require that a nurse should be able to access records of only those patients who are admitted to the ward where the nurse is currently located. This may be satisfied by requiring that the database service grants access for a patient's record to a *Nurse* role member only if that nurse is currently present in the patient's ward. We call such constraints *resource access constraints*. In the literature, similar requirements have been identified and addressed as part of the role graph model [15]. The *parameterized roles* in that model are similar to the *resource access constraints* in our model.

We observe that the role permission activation mechanism presented in Section 2.5 is inadequate for specifying such constraints requiring fine-grained access control of resources managed by a service. This is because of the following reasons. First, the specific resource to which access needs to be granted may not be known a priori. The resource is only identified at runtime, based on dynamic context information. Second, using permission activation to enforce such access control requirement may also lead to information leakage. A role member would be able to find out information about a resource's attributes simply through the failure or the success of the access attempt without ever requiring to access the resource.

3. CONTEXT-AWARE RBAC MODEL

In Figure 1 we present the elements of the CA-RBAC model. We distinguish between the context management layer and the access control layer.

3.1 Context management layer

Expressiveness of the context-based constraint specifications as part of the access control layer depends on the context models that are defined by the application. It is the responsibility of the application designer to define the appropriate context models based on the application requirements. Design of such models also depends on the available sensing technologies. For example, a nurse's location may be modeled at the granularity of a ward or based on the proximity of the nurse to a specific patient in a ward. The available location tracking sensors would determine this granularity.

The purpose of context models is to drive the design of the context management services for aggregating sensor data to generate context information required by an application. In the literature different approaches have been used for context modeling [31]: These include attribute-value pairs to represent context elements [29], domain specific ontologies, such as RDF and OWL [35], and the graphical approach involving Object-Role Modeling (ORM) framework [18, 19]. Attribute-value pairs are easy to handle but provide limited functionality. XML schemas support interoperability and shared understanding of a context model among different consumers of the context information. Ontologies provide mechanisms to define relationships between different context elements and domain concepts. The ORM approach [19] provides graphical mechanisms that aid in the development of context models for the domain of interest.

Ambient context detection requires continuous sensing of various different kinds of conditions in the environment, possibly at different locations, and real-time aggregation of the continuous streams of sensor data to infer the context conditions of interest. Application specific processing of this sensor data may be required to derive high-level context information. Applications may create and install one or more *context agents* for this purpose. The context agents need to authenticate the sensors from which they would collect sensor information. An application needs to trust the context agents from which it obtains the required context information because it is used in making access control decisions. Similar to our notion of context agents other researchers have developed different programming abstractions for accessing context information. These include context widgets [27], situations [19], and sentient objects [14].

In addition to ambient context, an application may also utilize *internal* context information in making access control decisions. Internal context corresponds to role membership related information and history of operation executions by role members. The role managers and the object managers provide interfaces to access this information.

In our model, context conditions are expressed as *predicates* that are evaluated by role/object managers. Such predicates are composed of queries supported by context agents and role managers. These predicates are used by the role/object managers as part of context-based access control policy enforcement. Occurrence of certain context conditions is communicated by the context agents to the access control layer through the notification of context events.

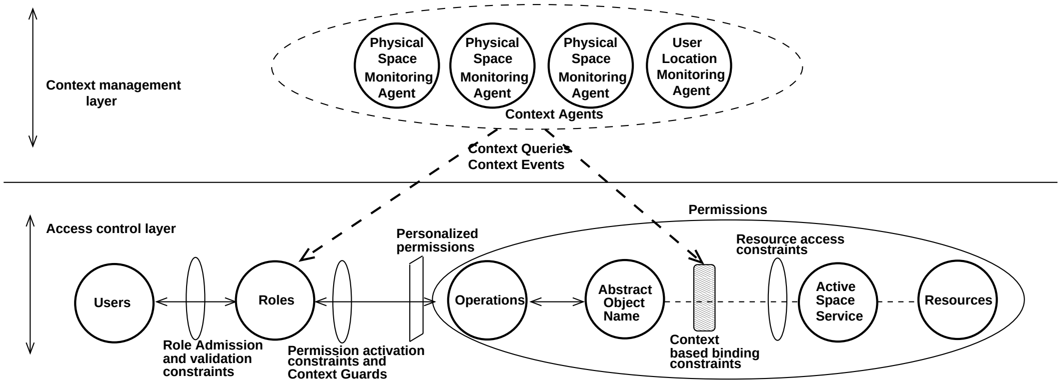
Context management layer
Physical Space Monitoring Agent
Physical Space Monitoring Agent
Physical Space Monitoring Agent
User Location Monitoring Agent
Context Agents
Context Queries
Context Events
Access control layer
Permissions
Personalized permissions
Resource access constraints
Users
Roles
Operations
Abstract Object Name
Active Space Service
Resources
Role Admission and validation constraints
Permission activation constraints and Context Guards
Context based binding constraints

Figure 1: Context-Aware RBAC Model (CA-RBAC)

3.2 Access control layer

One major distinction between our CA-RBAC model and the NIST RBAC model lies in the concept of *permissions* as considered in the two models. In the NIST RBAC model a permission specifies approval to perform specified actions on a particular object. The effect of a permission execution by different role members is always the same; it does not differ for different role members. Also, the objects on which permissions are specified are *statically* known in the system.

Our definition of a *permission* differs from that of the NIST RBAC model as follows. First, permissions may be *personalized* for each role member. A role operation when executed by different role member may access different objects based on the context of the individual role members. Second, permissions are specified on *objects* that may bind to different active space services under different context conditions.

To support *personalized permissions*, objects in our model are classified as *shared* and *private*. Shared objects are common to all members of all the roles, whereas private objects are specific to a role and are managed separately for different members of that role. The bindings of both shared and private objects may change based on context conditions. Personalized permissions are specified as role operations on the private objects defined in a role. The objects accessed by a role member through the personalized permissions belong to that member's private object space.

A *resource access constraint* can be associated with a role operation to enforce context-based access to resources managed by the service that is being accessed as part of that role operation. Such a constraint is a context-based predicate specifying a relation between a resource's attributes and the context variables associated with the role member invoking the operation. Only those resources that satisfy this predicate are allowed to be accessed through the role operation. The object manager that binds to the service evaluates these constraints in its interactions with that service. As part of evaluating these constraints, it utilizes the context information associated with the role member invoking the role operation.

To support context-based permission activation, the model provides the *precondition* mechanism. A precondition can be associated with a role operation. It consists of predicates involving queries to context agents, and operation event count based predicates [2]. It is evaluated by the role manager before the corresponding operation is allowed to be executed.

For addressing the *context invalidation problem*, we provide the *context guard* mechanism in our model. A context guard may be associated with each role operation individually. It consists of a context predicate that is required to remain valid while a role member's interactive session with the service that is being accessed through that role operation is active. The role manager evaluates the context guard and terminates the interactive session if the context predicate fails to hold.

4. PROGRAMMING FRAMEWORK

Based on the CA-RBAC model we have designed and implemented an XML-based programming framework for building context-aware pervasive computing applications [33]. In this framework, a context-aware application is programmed using an abstraction called *activity*. An activity may be distributed across different active spaces and it may involve multiple users in some collaborative tasks.

An activity provides the *role* abstraction. Objects that are defined in the activity are *shared* by all the members of all the roles, whereas objects that are defined within each role are *private* to that role. An object can be specified to bind to some service in the active space or to some context agent. Within an activity we do not distinguish between objects that refer to context agents and those referring to other services. This allows us to treat all the objects in a uniform way. However, there are two consequences of this decision. First, within an activity we need to specify an *order* for binding the objects. It is crucial to bind the objects referring to context agents *before* binding other objects in the activity for an application's correct behavior. In the programming framework we provide a construct for this purpose. Second, based on a given specification, the underlying middleware needs to grant permissions to the various role managers for accessing the context agents as part of role operation precondition evaluations.

The middleware provides three generic components, an *activity manager*, a *role manager*, and an *object manager*. The runtime environment of an application is constructed by specializing these managers based on the application's specification. All the managers are executed on a set of trusted servers in the active space. For each user, an interface component called *User Coordination Interface*

(UCI) is dynamically created and transported to that user's device. Through the UCI a user contacts a role manager for executing a role operation.

A separate object manager is created for each object defined in the activity. Similarly, for objects defined within a role, separate object managers are created corresponding to each user admitted to the role. An object manager maintains a reference to the service to which it is currently bound. It enforces context-based binding policies. It also enforces the resource access constraints that are supplied to it by the role manager as part of role operation invocation.

A separate role manager is created for each role defined in the activity. It enforces role admission and validation constraints, operation preconditions, and context guards. Role validation constraints and operation preconditions are evaluated every time a user invokes a role operation. An operation's actions are performed only if the operation precondition and the role validation constraints are satisfied.

As part of an operation execution, the role manager contacts the object manager corresponding to the object on which the operation is specified to be executed. The object manager invokes on the currently bound service the methods that are specified as part of the role operation. For monitoring and gathering external context information we use an agent-based distributed event monitoring system [34].

We now illustrate our programming framework by presenting how access control requirements for the two applications presented in Section 2 can be programmed in our framework. For both the applications we used the following experimental setup. We defined *room agents* corresponding to various rooms in our department building. Each such agent maintained status information about its room, such as the list and the number of users in the room. Users' Bluetooth enabled devices, such as laptops, and PDAs, are used to detect user presence in a room. The agent generated the *StatusChangeEvent* when a person arrived or left the room. A user location monitoring agent was defined in the testbed environment to maintain location information for each user. In the activity we defined a *LocationServiceAgent* object to refer to this agent. This agent subscribed to the *StatusChangeEvent* from each room agent in the system. It generated *LocationChangeEvents* corresponding to each user as he/she moved from one room to another.

4.1 Role admission and validation

The programming framework supports two constructs for programming role admission and validation constraints that are specified with each role. The specified constraints are evaluated by the corresponding role manager. Validation constraints are evaluated every time a role member attempts to execute a role operation. Below we illustrate the usage of these constructs as part of the patient information activity.

We consider the following two requirements related to a user's memberships in a nurse-on-duty role associated with a ward. First, we require that a user should be admitted to the nurse-on-duty role corresponding to a ward only if he/she is also a member of the *Nurse* role. Second, we require that the user's membership in the nurse-on-duty role of a ward should be revoked if the user goes out of that ward, or after his/her time of duty is over, or if the user's membership in the *Nurse* role is revoked.

Below we present the partial specification of the *PatientInformationSystem* activity, enforcing these two requirements. We use a pseudo notation for presenting the specification examples. In this notation the **boldface** terms represent XML tags in our programming framework. We define the *NurseOnDutyEmergencyWard* role and the *EmergencyWardAgent* object in this activity. We bind the object to a specific ward's context agent at the activity instantiation time.

```
Activity PatientInformationSystem {
   Role NurseOnDutyEmergencyWard {
      AdmissionConstraint { member(thisUser, Nurse) }
      ValidationConstraint {
         EmergencyWardAgent.isPresent(thisUser) &&
         current_time <= DATE (Mar, 21, 2008, 10:00) &&
         member(thisUser, Nurse)
      }
   } // end of role
   Object EmergencyWardAgent {
      Bind Direct (//WardAgentURL)
   } // end of object
}
```

The first requirement is specified through the *AdmissionConstraint* construct. As part of this constraint we check for the user's membership in the *Nurse* role. The function *member(thisUser, RoleName)* is provided in the programming framework to check the invoking users membership in the role *RoleName*. The variable *thisUser* is a special variable in the programming framework which translates to the identifier of the role member who is requesting admission to the role.

The second requirement is specified through the *ValidationConstraint* construct. As part of this constraint the role manager queries the *EmergencyWardAgent* object to check whether this role member is present in the ward. If the role member is not present in the ward, or if the current time is past the specified time, or if the user is no longer a member of the *Nurse* role then the user's membership in the *NurseOnDutyEmergencyWard* role is revoked.

4.2 Context-based object bindings

A context-based permission is programmed as a role operation on an object which may bind to different active space services under different context conditions. For programming an object's context-based binding policies, the *Reaction* construct is provided in the programming framework. One or more reactions may be specified with an object definition. All such reactions are handled by the corresponding object manager. A reaction follows the Event-Condition-Action (ECA) model of evaluation. It is triggered by one or more *context events*. The condition specification consists of queries to context agents. The action consists of the binding specification for the object.

In order to illustrate the context-based object binding and the use of the *Reaction* construct, we consider the context-based requirements of the music player application. In this application we require that when a user enters a room, the application is able to access the audio player service in that room only if no other user is present in the room. In Figure 2 we present a partial specification of this activity that only addresses the above requirement. Other requirements presented in Section 2 for this application are not considered here.

We define two objects, *CurrentRoom* and *AudioPlayer*, within the *User* role. Both these are *private* objects of this role. Through the *BindingOrder* construct we specify that

```
Role User {
    BindingOrder { CurrentRoom AudioPlayer }
    Object CurrentRoom RDD (//RoomRDD.xml) {
        Reaction {
            When Event LocationChangeEvent(thisUser)
                Bind Discover
                (LOCATION=LocationServiceAgent.getLocation
                    (thisUser))
        } // end of binding reaction
    } // end of object definition
    Object AudioPlayer RDD (//AudioPlayerRDD.xml) {
        Reaction {
            When Event LocationChangeEvent(thisUser)
            Precondition CurrentRoom.isPresent(thisUser) &&
                CurrentRoom.presentUserCount() == 1
                Bind Discover
                (LOCATION=LocationServiceAgent.getLocation
                    (thisUser))
        } // end of reaction
    } // end of object definition
    Operation PlayMusic {
        Precondition AudioPlayer.isBound()
            Action AudioPlayer.play()
    } // end of operation
} // end of User Role
```

Figure 2: Context-based object binding example

the *CurrentRoom* object should be bound *before* binding the *AudioPlayer* object. This is crucial for the correct execution of this activity because the *CurrentRoom* object is accessed as part of the condition evaluation of the *AudioPlayer* object's binding reaction.

The binding reaction of the *CurrentRoom* object is triggered by the *LocationChangeEvent* corresponding to the user. The object is bound by *discovering* the room agent based on the role member's location. Such a discovery is performed as follows. Associated with each object is a XML description called RDD (Resource Description Definition), which specifies the attributes and interfaces of the service to which that object may be bound. We use RDD for service discovery in our system. Some of the attributes' values in a RDD can be based on the context information at the discovery time. In the above example, the *LOCATION* attribute value in the *RoomRDD* is filled in at runtime by querying the location of the role member from the *LocationServiceAgent*.

For the *AudioPlayer* object we specify a reaction that binds the object to the audio player service in the room where the user is present. This reaction is triggered by the *LocationChangeEvent* corresponding to the user. As part of the reaction's condition, the *AudioPlayer* object manager queries the room agent to which the *CurrentRoom* object is bound. The condition checks whether this role member is the only person present in the room. The binding action is performed if this condition is true. The discovery procedure is same as the binding of the *CurrentRoom* object. The user may initiate playing of music through the *PlayMusic* operation.

4.3 Personalized role permissions

A personalized role permission is programmed as a role operation on a *private* object defined in a role. We illustrate this through the following example. In the patient information system we want that every member of the *Nurse* role should be able to access the printer service in the ward where that member is located. In Figure 3 we show how this requirement is programmed in our framework. In the *Nurse* role we define the *Print* operation through which the private *MyPrinter* object may be accessed.

```
Role Nurse {
    Object MyPrinter RDD (//PrinterRDD.xml) {
        Reaction { ... }
    } // end of MyPrinter object definition
    Operation Print {
        Action MyPrinter SessionMethod print
    } // end of operation
} // end of Nurse Role
```

Figure 3: Personalized role permissions

The *MyPrinter* object refers to separate printer service instances for each role member. Therefore, the binding of this object is maintained separately and independently for each *Nurse* role member. The binding action for this object is similar to the binding of the *CurrentRoom* object presented in Figure 2.

In the programming framework the *SessionMethod* construct is provided to specify the list of methods that are allowed to be invoked as part of an *interactive session*. We observe that such an interactive session is used only for a role member's interaction with a service. It should not be confused with the concept of a *session* in the NIST RBAC model.

4.4 Permission activation and context guard

The context-based permission activation constraints are programmed as a *precondition* of a role operation. A role manager evaluates an operation's precondition before the operation's actions can be executed.

```
Role Nurse {
    Object CurrentWard { ... }
    Operation AccessCriticalReports {
        Precondition
            CurrentWard.isPresent(thisUser) &&
            CurrentWard.isPresent(members(Doctor))
        Action PatientDB
            SessionMethod accessDoctorReport
        ContextGuard {
            When StatusChangeEvent
            GuardCondition
                CurrentWard.isPresent(thisUser) &&
                CurrentWard.isPresent(members(Doctor))
        } // end of Context Guard
    } // end of role operation
} // end of Nurse role
```

Figure 4: Operation precondition and context guard example

In the patient information system we require that a *Nurse* role member may access doctor's patient reports only if a *Doctor* role member is also present in the ward. This requirement is programmed using the precondition construct as shown in Figure 4. We define the private object named *CurrentWard* in the *Nurse* role. For each nurse it is bound to the room agent corresponding to the room where that nurse is present. The object's binding changes based on the current location of a nurse. In this regard it is similar to the binding of the *CurrentRoom* object presented in Figure 2.

We define the *AccessCriticalReports* operation through which a *Nurse* role member may access doctor reports. As part of the operation precondition the *Nurse* role manager checks if the *Nurse* role member who is invoking the operation and some member of the *Doctor* role are present in the *CurrentWard*. The role manager executes the operation's actions only if the precondition is true. Execution of this operation leads to the initiation of an interactive session with the *database service*. As part of this session the role member may invoke *accessDoctorReport* method on the *PatientDB* object. We want that this session be terminated when either the nurse leaves the ward or when no member of the *Doctor* role remains in the ward.

In our programming framework we provide the *ContextGuard* construct for addressing the above kind of context invalidation problem. Such a construct identifies two things - *what* context predicate to evaluate, and *when* to evaluate it. In our framework, we use *context events* to trigger the evaluation of a context guard. A context guard becomes effective when the corresponding operation is executed. Once effective, the role manager evaluates the context condition specified through the *GuardCondition* construct every time the guard's evaluation is triggered by the notification of the specified context events. The role manager terminates the interactive session if the context condition fails to hold.

We specify a context guard for the *AccessCriticalReports* operation. Its evaluation is triggered by the *StatusChangeEvent* that is notified to the *Nurse* role manager by the ward agent to which the *CurrentWard* object is bound. The *Nurse* role manager terminates the interactive session if either the nurse has left the ward or if no member of the *Doctor* role is present in the ward.

4.5 Resource access constraint

A resource access constraint may be specified separately for every object invocation within a role operation. We define the *AccessConstraint* construct for this purpose in the programming framework.

```
Role Nurse {
   Operation AccessWardPatientInfo {
      Action PatientDB
         SessionMethod accessPatientInformation
         AccessConstraint
            (WardID =
            LocationServiceAgent.getLocation(thisUser))
   }
} // end of Nurse role
```

Figure 5: Resource access constraint example

In the patient information system we require that a *Nurse* role member may access the records of only those patients who are admitted to the ward where the nurse is currently present. This requirement is programmed as shown in Figure 5. In the *Nurse* role we define the *AccessWardPatientInfo* operation through which a role member may access patient records. Through the *AccessConstraint* construct we specify the required resource access constraint. It is enforced by the *PatientDB* object manager. The constraint specification consists of the *WardID* attribute of patient records. The *PatientDB* object manager grants access to only those database records for which the *WardID* attribute has the value equal to the location of the *Nurse* role member who is invoking the operation. For this, it queries the *LocationServiceAgent* to obtain the location of the nurse.

5. RELATED WORK

Other researchers have also developed RBAC models specifically for context-aware pervasive computing applications [28, 9, 24]. Gaia [28] defines three different role categories, corresponding to system-wide roles, active space roles, and application roles, and a mapping between them. In GRBAC model [9] context information is considered as the *environmental role*, which an application needs to posses in order to perform context-dependent tasks. Such a definition leads to large number of roles in an access control system, as there might be potentially many environmental states that are relevant for an application. In [24], context-based constraints are associated with activation of role permissions. They also provide engineering guidelines for building context-based RBAC systems.

The context-based access control mechanisms in our programming framework differ from the above systems in the following four ways. First, we provide role validation constraints to support revocations of role membership if context conditions fail to hold. Second, we support personalized role permissions that are executed on different objects for different role members. Third, we provide the *context guard* mechanism to revoke interactive sessions when context conditions fail to hold. Fourth, through the *resource access constraint* mechanism we enforce access control requirements that depend on the relationship of a resource's attributes with the context information, such as the identity and the location of the role member who is accessing the resource. Similar requirements have been discussed and addressed previously in [17, 15]. The mechanisms of *parameterized privileges* [17] and *parameterized roles* [15] essentially perform access control based on an object's contents. Content-based access control ideas can be traced back to the work on access control based on *data types* in programming languages [20].

Constraints specification as part of RBAC models have been extensively studied by researchers [3, 10]. We focus on context-based constraint specification as part of designing context-aware applications. This requires close interaction between the access control system and the context management layer. In our model, application specific context agents are deployed in the system to collect context information and generate context events. The context predicates are evaluated by the application specific role/object managers. A generic framework for context evaluation has been developed as part the Antigone system [22]. It provides mechanisms to enforce security of the context information used as part of authorization policies. In contrast, our work considers integration of context-based constraints in a RBAC model for pervasive computing applications. This leads to a number of requirements, discussed in this paper, which are not addressed by the Antigone framework.

Context-based constraints that limit a role's visibility to specific geographic areas are presented as part of the GEO-RBAC model [6]. Similarly, the GTRBAC model [21] provides mechanisms for enabling and disabling of roles based on temporal constraints. In our model context-based constraints including temporal and spatial constraints, are specified as part of role admission/validation and role operation preconditions. We argue that this approach supports fine-grained access control requirements, as one can selectively revoke a user's membership from a role, or activate/deactivate specific role permissions, instead of enabling/disabling a role. Moreover, we also support fine-grained access control through *resource access constraints* that are specified as part of a role operation. Additionally, we also provide the context guard mechanism to revoke role operation sessions when specified context conditions fail to hold.

The UCON model [26] provides an extensive framework to model a broad range of *usage control* policies. In contrast, we focus on access control for context-aware applications. There are conceptual similarities between some of the mechanisms in our programming framework and some of the UCON modeling abstractions. For instance, the context guard mechanism in our framework is similar to the UCON's decision predicates that are evaluated during a request execution (*ongoing-authorizations*). Moreover, the UCON's attribute-based access control mechanisms can be used to specify the requirements that are addressed by the *resource access constraints* in our framework. Support for attribute-based access control is also provided by the XACML standard for distributed authorization management [23].

The *resource access constraint* mechanism in our programming framework is similar to the *parameterized roles* of the role graph model [15] and *parameterized privileges* of [17]. Additionally, we also provide the mechanism of *personalized role permissions* in our model. The personalized role permissions and the resource access constraint mechanism serve different purposes. Personalized role permissions (private object space for a role member) provide a mechanism to control a role member's access to a service based on his/her context information. Resource access constraints on the other hand provide a mechanism to control a role member's access to a subset of resources managed by a service. A resource access constraint provides a finer granularity of access control as compared to the personalized role permissions.

Distinction between an object type and its instance for access control has also been considered in the TMAC model [32]. In contrast to that model, the abstract objects in our model may be dynamically bound to different services under different context conditions. This requires object managers to verify the authenticity of such services before binding. Such a requirement does not arise in the TMAC model, making its implementation possibly simpler than our model.

6. CONCLUSIONS

In this paper we have demonstrated the need for extending the NIST RBAC model for addressing context-based access control requirements of pervasive computing applications. We presented our context-aware RBAC model that supports several such extensions. In this model we distinguish between the context management layer and the access control layer. The model supports personalized permissions for role members, and context-based constraint specification as part of - dynamic binding of objects with active space services, user admission to roles, permission executions by role members, and granting access to a subset of a service's resources based on a role member's context information. The model also supports revocation of a user's membership in a role when context conditions fail to hold. We have also identified the *context invalidation problem* in this paper. To address this problem we provide the context guard mechanism in our CA-RBAC model. Based on this model we have developed a role-based framework for programming secure context-aware pervasive computing applications. In this paper we demonstrated the key elements of this framework through a set of application examples that we have implemented as part of our testbed suite.

7. ACKNOWLEDGMENTS

We are thankful to Prof. Sylvia Osborn and all the reviewers for providing valuable feedback that resulted in improving the paper's presentation.

8. REFERENCES

[1] G. D. Abowd, A. K. Dey, P. J. Brown, N. Davies, M. Smith, and P. Steggles. Towards a Better Understanding of Context and Context-Awareness. In *HUC '99: Proceedings of the 1st International Symposium on Handheld and Ubiquitous Computing*, pages 304–307. Springer-Verlag, 1999.

[2] T. Ahmed and A. R. Tripathi. Specification and Verification of Security Requirements in a Programming Model for Decentralized CSCW Systems. *ACM Transactions on Information and System Security (TISSEC)*, 10(2):7, 2007.

[3] G.-J. Ahn and R. Sandhu. Role-based Authorization Constraints Specification. *ACM Transactions on Information and System Security (TISSEC)*, 3(4):207 – 226, November 2000.

[4] J. Bacon, K. Moody, and W. Yao. A Model of OASIS Role-based Access Control and its support for Active Security. *ACM Transactions on Information and System Security (TISSEC)*, 5(4):492–540, 2002.

[5] J. E. Bardram, T. R. Hansen, M. Mogensen, and M. Søgaard. Experiences from Real-World Deployment of Context-Aware Technologies in a Hospital Environment. In *Ubicomp*, pages 369–386, 2006.

[6] E. Bertino, B. Catania, M. L. Damiani, and P. Perlasca. GEO-RBAC: A Spatially Aware RBAC. In *SACMAT '05: Proceedings of the Tenth ACM Symposium on Access control Models and Technologies*, pages 29–37, 2005.

[7] R. Campbell, J. Al-Muhtadi, P. Naldurg, G. Sampemane, and M. D. Mickunas. Towards Security and Privacy for Pervasive Computing. In *Lecture Notes in Computer Science Software Security - Theories and Systems*, volume 2609, pages 77–82. Springer, 2003.

[8] S. Consolvo, P. Roessler, B. E. Shelton, A. LaMarca, B. Schilit, and S. Bly. Technology for Care Networks of Elders. *IEEE Pervasive Computing*, 3(2):22–29, 2004.

[9] M. J. Covington, W. Long, S. Srinivasan, A. K. Dey, 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, 2001.

[10] J. Crampton. Specifying and Enforcing Constraints in Role-based Access Control. In *SACMAT '03: Proceedings of the Eighth ACM Symposium on Access control Models and Technologies*, pages 43–50, 2003.

[11] N. Davies, K. Cheverst, K. Mitchell, and A. Efrat. Using and Determining Location in a Context-sensitive Tour Guide. *IEEE Computer*, 34(8):35–41, August 2001.

[12] M. Evered and S. Bögeholz. A Case Study in Access Control Requirements for a Health Information System. In *ACSW Frontiers '04: Proceedings of the Second Workshop on Australasian Information Security, Data Mining and Web Intelligence, and Software Internationalisation*, pages 53–61, 2004.

[13] D. F. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramouli. Proposed NIST standard for Role-based Access Control. *ACM Transactions on Information and System Security (TISSEC)*, 4(3):224–274, 2001.

[14] A. Fitzpatrick, G. Biegel, S. Clarke, and V. Cahill. Towards a Sentient Object Model. In *Workshop on Engineering Context-Aware Object-Oriented Systems and Environments (ECOOSE)*, November 2002.

[15] M. Ge and S. L. Osborn. A Design for Parameterized Roles. In *DBSec*, pages 251–264, 2004.

[16] 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, 2001.

[17] L. Giuri and P. Iglio. Role Templates for Content-based Access Control. In *RBAC '97: Proceedings of the Second ACM Workshop on Role Based Access Control*, pages 153–159, 1997.

[18] T. Halpin. *Information Modeling and Relational Databases: From Conceptual Analysis to Logical Design*. Morgan Kaufmann Publishers Inc., 2001.

[19] K. Henricksen and J. Indulska. A Software Engineering Framework for Context-Aware Pervasive Computing. In *PERCOM '04: Proceedings of the Second IEEE International Conference on Pervasive Computing and Communications (PerCom'04)*, page 77, 2004.

[20] A. K. Jones and B. H. Liskov. A Language Extension for Expressing Constraints on Data Access. *Commun. ACM*, 21(5):358–367, 1978.

[21] J. B. D. Joshi, E. Bertino, U. Latif, and A. Ghafoor. A Generalized Temporal Role-Based Access Control Model. *IEEE Transactions on Knowledge and Data Engineering (IEEE TKDE)*, 17(1):4–23, 2005.

[22] P. McDaniel. On Context in Authorization Policy. In *SACMAT '03: Proceedings of the Eighth ACM Symposium on Access control Models and Technologies*, pages 80–89, 2003.

[23] T. Moses. OASIS eXtensible Access Control Markup Language (XACML) Version 2.0, OASIS Standard. pages 1–141, 1 February 2005.

[24] 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, 2003.

[25] U. Nitsche, R. Holbein, O. Morger, and S. Teufel. Realization of a Context-Dependent Access Control Mechanism on a Commercial Platform. In *Proceedings of IFIP/SEC 1998*. Chapman & Hall.

[26] J. Park and R. Sandhu. The $UCON_{ABC}$ Usage Control Model. *ACM Transactions on Information and System Security (TISSEC)*, 7(1):128–174, 2004.

[27] D. Salber, A. K. Dey, and G. D. Abowd. The Context Toolkit: Aiding the Development of Context-Enabled Applications. In *Proceedings of the 1999 Conference on Human Factors in Computing Systems (CHI '99)*, pages 434–441, May 1999.

[28] G. Sampemane, P. Naldurg, and R. H. Campbell. Access control for Active Spaces. In *Annual Computer Security Applications Conference (ACSAC2002)*, 2002.

[29] B. Schilit, N. Adams, and R. Want. Context-Aware Computing Applications. In *IEEE Workshop on Mobile Computing Systems and Applications*, pages 85–90, Santa Cruz, CA, US, 1994.

[30] Y. Shi, W. Xie, G. Xu, R. Shi, E. Chen, Y. Mao, and F. Liu. The Smart Classroom: Merging Technologies for Seamless Tele-Education. *IEEE Pervasive Computing*, 02(2):47–55, 2003.

[31] T. Strang and C. Linnhoff-Popien. A Context Modeling Survey. In *Workshop on Advanced Context Modelling, Reasoning and Management as part of UbiComp 2004 - The Sixth International Conference on Ubiquitous Computing*, September 2004.

[32] 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, 1997.

[33] A. Tripathi, D. Kulkarni, and T. Ahmed. A Specification Model for Context-Based Collaborative Applications. *Elsevier Journal on Pervasive and Mobile Computing*, 1(1):21 – 42, May-June 2005.

[34] A. R. Tripathi, D. Kulkarni, H. Talkad, M. Koka, S. Karanth, T. Ahmed, and I. Osipkov. Autonomic Configuration and Recovery in a Mobile Agent-based Distributed Event Monitoring System. *Software - Practice & Experience*, 37(5):493–522, 2007.

[35] X. H. Wang, D. Q. Zhang, T. Gu, and H. K. Pung. Ontology Based Context Modeling and Reasoning Using OWL. In *PERCOMW '04: Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications Workshops*, 2004.