2011

Prox-RBAC: A Proximity-based Spatially Aware RBAC

Michael Kirkpatrick

Maria Luisa Damiani

Elisa Bertino

Purdue University, bertino@cs.purdue.edu

Follow this and additional works at: http://docs.lib.purdue.edu/ccpubs

Part of the Engineering Commons, Life Sciences Commons, Medicine and Health Sciences Commons, and the Physical Sciences and Mathematics Commons


http://docs.lib.purdue.edu/ccpubs/395

This document has been made available through Purdue e-Pubs, a service of the Purdue University Libraries. Please contact epubs@purdue.edu for additional information.
Prox-RBAC: A Proximity-based Spatially Aware RBAC

Michael S. Kirkpatrick*  
Department of Computer Science  
Purdue University  
West Lafayette, IN, USA  
mkirkpat@cs.purdue.edu

Maria Luisa Damiani  
Department of Informatics and Communication  
University of Milan, Italy  
damiani@dico.unimi.it

Elisa Bertino  
Department of Computer Science  
Purdue University  
West Lafayette, IN, USA  
bertino@cs.purdue.edu

Categories and Subject Descriptors  
H.2.8 [Database Management]: Database Applications—Spatial Databases and GIS; K.6.5 [Computing Milieus]: Security and Protection

General Terms  
Design, Languages, Security, Theory

Keywords  
access control, security, gis, mobility

ABSTRACT  

As mobile computing devices are becoming increasingly dominant in enterprise and government organizations, the need for fine-grained access control in these environments continues to grow. Specifically, advanced forms of access control can be deployed to ensure authorized users can access sensitive resources only when in trusted locations. One technique that has been proposed is to augment role-based access control (RBAC) with spatial constraints. In such a system, an authorized user must be in a designated location in order to exercise the privileges associated with a role. In this work, we extend spatially aware RBAC systems by defining the notion of proximity-based RBAC. In our approach, access control decisions are not based solely on the requesting user’s location. Instead, we also consider the location of other users in the system. For instance, a policy in a government application could prevent access to a sensitive document if any civilians are present. We introduce our spatial model and the notion of proximity constraints. We define the syntax and semantics for the Prox-RBAC language, which can be used to specify these policy constraints. We introduce our enforcement architecture, including the protocols and algorithms for enforcing Prox-RBAC policies, and give a proof of functional correctness. Finally, we describe our work toward a Prox-RBAC prototype and present an informal security analysis.

1. INTRODUCTION  

Role-based access control (RBAC) has been widely adopted as a way to streamline the maintenance of access control policies. As the deployment of mobile devices in these settings has increased, researchers have explored techniques for augmenting RBAC with contextual information. Perhaps the most common extension of RBAC is to incorporate the location constraints [9, 17, 14, 1, 20]. A common form of restriction is to constrain the use of roles to specific geographic locations. Suppose for example that an organization has a role Manager and this role is constrained to the location Office. A user wishing to use this role must not only have the authorization to use this role, but also be in his office when using the role. Consequently, the user would be prevented from using the role and the permissions granted to the role from other locations.

A major limitation of current approaches to spatially aware RBAC is that they focus only on the location of the user issuing the role usage request. However in many real situations whether a user can use a role and access the resources for which access is granted to the role may depend on the presence or absence of other users. As an example, consider a government agency with data classified at multiple levels of security. One policy could prohibit access to a sensitive document if there are any civilians (i.e., non-governmental employees) present. Another could require the presence of a supervisor when a document is signed. Yet another could require that the subject is alone (e.g., “for your eyes only” restrictions).

In this paper, we address the problem of specifying and enforcing a novel class of location constraints, referred to as proximity-based location constraints, for RBAC for both static and mobile environments. That is, we want to make decisions about granting access to roles by also taking into account the location of other users, possibly considering the proximity of those users to the requesting subject. Incorporating contextual factors in mobile environments is challenging, as these environments are inherently dynamic. As such, it is important to consider how to monitor and react to changes in users’ locations. This challenge can be described as enforcing continuity of usage constraints. Our approach is to adopt the policy language semantics of the UCON family of access control models [25, 18]. This family of models defines a number of semantic structures that enable the specification of contextual access control policies.

To illustrate the integration of continuity of usage with proximity-based constraints, consider the movement of a user from one protected region to another. For this user, it would be desirable to preserve the existing permissions in use if possible. However, the movement may involve entering a region in which a spatial role is not permitted to be used. Consequently, the movement would trigger a policy re-evaluation, revoking permissions as needed. In addition, the movement may impact another user by violating a
condition of the latter’s policy constraints. Another challenge is to bind the user to the device making the request, as well as the location. That is, it would be undesirable for the user to download sensitive data to a device, then proceed to allow another user to leave with the device (and the data).

To address these challenges, we propose Prox-RBAC, which consists of a model, policy language, and enforcement architecture for specifying and enforcing proximity-based location constraints. In Prox-RBAC, administrators can write policies that specify either the presence or absence of other users within a protected area. Prox-RBAC also makes a distinction between policies that require authorization only a single time prior to access and policies that specify conditions that must continue to hold for as long as the permission is used. Finally, Prox-RBAC is backward compatible; that is, Prox-RBAC can specify existing spatially aware RBAC policies, as well as traditional RBAC. Thus, the contributions of this work can be summarized as follows.

- We introduce the notion of proximity constraints and provide a formal definition of these constraints.
- We specify the syntax and semantics of Prox-RBAC, a language for expressing proximity constraints in a spatial RBAC system.
- We propose the integration of proximity constraints with continuity of usage within the context of a constrained indoor space model.
- We define an enforcement architecture, including the specification of a cryptographic protocol and algorithms for Prox-RBAC.

2. BACKGROUND

Prox-RBAC builds on the GEO-RBAC spatially aware RBAC model. In addition, we use the UCONABC family of models to define the semantics of the Prox-RBAC language. In this section, we provide a brief summary of these existing works.

2.1 GEO-RBAC

Prox-RBAC is defined as an extension of GEO-RBAC, a spatially aware RBAC model. In GEO-RBAC, traditional RBAC roles are augmented to incorporate the subject’s location. Policies can use these spatial roles in defining fine-grained access control permissions. For instance, policies using the spatial role < Manager, Room 513 > would require that the user activate the Manager role (assuming the user is authorized) and be physically present in room 513. Consequently, a subject using the role Manager would be denied access if the request is made from another location.

A key feature of GEO-RBAC that we use in Prox-RBAC is the differentiation of role enabling and role activation. A spatial role is automatically enabled if the user is authorized to activate the role and the user is physically present in the requisite location. However, an enabled role does not explicitly grant any privileges. Instead, the user must activate the role in order to exercise the associated permissions. This differentiation allows for mutually exclusive roles and hierarchical roles to be defined on the same space. It is the user’s specific action that determines the role applied.

2.2 UCONABC

UCONABC is a family of access control models that can be used to formalize the behavior of a system in terms of authorizations (A), obligations (B), and conditions (C), that must be satisfied either before (pre), during (on), or after (post) an access occurs. For instance, UCONpreA can be used to formalize an access control system that requires authorizing the subject before the access is granted. With each type of model, there are multiple variations (e.g., UCONpreA, UCONpostA, and UCONpreA). For a full description of these details, we refer the reader to the UCONABC paper [25, 18].

Specifying a policy in UCONABC can be done by declaring the functions, relations, and other mathematical structures, and then defining the implication that must hold if the access is granted. As an example, consider a traditional RBAC system, where S, O, A and R denote the sets of subjects, objects, actions, and roles, respectively. A permission is an ordered pair (o, a) for o ∈ O, a ∈ A. Subjects are mapped to active roles via the ActiveRoles function, while PermittedRoles maps permissions with the roles that are to be granted. As ActiveRoles are user-specific, we also declare ATT(S) = {ActiveRoles}, where ATT(S) specifies the set of attributes that are associated with subjects.

To specify that a subject s is authorized to perform action a on object o, UCONpreA uses an invariant allowed(s, o, a) → P, where P denotes a necessary condition for authorization. For instance, in RBAC, the necessary condition is that the requester has an active role (∃role ∈ ActiveRoles(s)) that is granted the desired permission (∃role′ ∈ PermittedRoles(o, a), role ≥ role′). The following example illustrates how UCONpreA can specify traditional RBAC policies.

To enforce that certain conditions continue to hold as the access occurs (i.e., continuity of usage constraints), UCONABC defines additional primitive formalisms. First, preCON declares a set of conditions that are evaluated, while getPreCON(s, o, a) specifies proposition built on these conditions. Then preConChecked can be used in the necessary proposition in the allowed(s, o, a) → P implication. For instance, if the subject must be over the age of 18 or accompanied by an adult, we could specify this portion of the policy as:

preCON = {Over18(s), Accompanied(s)}
getPreCON(s, o, a) = Over18(s) ∨ Accompanied(s)
allowed(s, o, a) ⇒ preConChecked(getPreCON(s, o, a))

For on-going conditions (i.e., UCONonC), similar structures exist (i.e., onCON and getOnCON(s, o, a)). Specifying permission revocation is similar to the allowed(s, o, a) invariant. Specifically, allowed(s, o, a) ⇔ ~onConChecked(getOnCON(s, o, a)) declares that the access must be stopped once the on-going required condition is no longer true.

3. THE PROX-RBAC LANGUAGE

In this section we present the syntax and semantics of our proximity constraint language. After a short preliminary subsection to introduce some notation, we describe our space model, which is a key element in the definition of the proximity constraint model. We then briefly introduce our spatial role model, followed by the introduction of the three main constructs of our proximity constraint model. Such introduction is followed by the definition of the syntax and semantics of the proximity constraint model.
3.1 Preliminaries

The core Prox-RBAC model, which underlies our language, uses a number of primitives that are similar to existing spatially aware RBAC models. As in traditional RBAC, we have Subjects ($S$) that can request permission to perform Actions ($A$) on Objects ($O$). Let Roles ($R$) denote the set of roles in the system. When $s \in S$ requests the privilege to perform $a \in A$ on $o \in O$, $s$ must activate a role $r \in R$ to which the requested permission is granted.

3.2 Space Model

The first challenge in defining our model is to specify how space is modeled. We adopt a spatial model that subdivides generic spaces into regions based on security classifications. We consider the reference space $\top$ to be a region that is divided into a set of protected areas (PA). A PA is a physically bounded region of space, accessible through a limited number of entry points, which consists of a physical barrier that requires authorization. Each PA can be arbitrarily large and we place no restrictions on the internal structure. For instance, a PA could consist of a single room or an entire floor that is made up of distinct but unlocked rooms. In the latter case, the subject’s presence in a particular room is irrelevant to the security questions, and we model the floor in its entirety as a PA. An entry point is a one-way walking device which connects one PA to another PA. Should the passage be in both directions, two distinct entry points are needed, one for each direction.

Figure 1 demonstrates a number of characteristics of our space. The PAs are distinguished by shading and lines filling the area. The red dots indicate the entry points and are labeled as the guard device $e_{i,j}$ controlling passage from $pa_i$ to $pa_j$. Observe that rooms 103A and 103B are both part of $pa_2$, thus indicating that PAs do not necessarily have a one-to-one correspondence to the features of the space. In addition, suite 100A defines $pa_3$ and consists of the entire space shaded with gray, which includes rooms 101, 102, 103A, and 103B. Similarly, $pa_2$ covers the entire floor. This illustrates that PAs can be defined hierarchically. Finally, $pa_6$ defines the outdoor space.

To represent this constrained space we define an indoor space model [16, 15]. Indoor space models present distinguishing features which perfectly match the requirements posed by our scenario. A major feature is that those spaces are cellular (or symbolic), i.e., they consist of a finite set of named cells or symbolic coordinates (e.g., rooms 103A and 103B) [3]. Moreover, indoor spaces may present complex topologies. Among the various topologies that one can specify, the most relevant are the connectivity between the indoor and outdoor spaces, as well as connectivity between cells (i.e., PAs). We choose to define the indoor space model as the tuple $(P, E, G, H)$ where $P$ is the set of PAs, $E$ the set of entry points, $G$ the connectivity topology referred to as accessibility graph, and $H$ the hierarchy of PAs. This space model allows us to introduce the notion of accessibility graph for a specific subject, defining the subspace in which the subject is authorized to move.

3.2.1 Hierarchy of PAs

As noted above, our model incorporates the notion of hierarchy of PAs. For instance, access to the first floor of a building may be restricted to a set of users; in addition, access to room 105 may be further restricted to an individual user. As such, when the user is in room 105, the permissions associated with the first floor should still be granted.

To model this hierarchy, we introduce the partial order $\sqsubseteq$ between PAs such that $pa_i \sqsubseteq pa_j$ (i.e., $pa_i$ is-part-of $pa_j$) means: i) any subject present in $pa_i$ is also present in $pa_j$; ii) $pa_i$ can be only entered if the subject is present in $pa_j$; iii) $pa_i$ is part of both $pa_j$ and $pa_k$.

Note that the meaning of the partial order relation is not simply of spatial containment. That motivates the second condition which explicitly requires the two PAs to be connected through an entry point. For instance, consider a room (Room 204) whose only entrance is a private stairwell from a room on the first floor; since this room is not reachable from any other part of the second floor, Room 204 $\not\sqsubseteq$ Floor 2. The third condition states that the partial order takes the form of a tree where the root is the reference space and the leaves are the PAs which do not contain further PAs. Note that, as $\sqsubseteq$ is a partial order, if $pa_i \sqsubseteq pa_j$, it cannot happen that $pa_k \sqsubseteq pa_j$, for $i \neq j$.

Therefore, the number of subjects in $pa_j$ is equal or greater than the sum of the subjects in the children of $pa_j$. This partial order leads naturally to the notion of type; for instance, if $pa_i$ is of the room type and $pa_j$ is the floor type, it is possible that $pa_i \sqsubseteq pa_j$, but it cannot happen that $pa_i \sqsubseteq pa_j$. In addition, we write $s \in pa_i$ to express the user’s location at the finest granularity. That is, if $s \in pa_i$, then $\not\in pa_j$, with $pa_i \sqsubseteq pa_j$, and $i \neq j$ such that $s \in pa_j$. Figure 2(b) illustrates the hierarchy corresponding to Figure 1.

3.2.2 Accessibility graph

The accessibility graph represents the relationships of connectivity among PAs through a directed multigraph (multidigraph). Figure 2 shows the accessibility graph for the space in Figure 1. The accessibility graph consists of a set $V$ of vertices each representing either the outdoor space or a PA, and a multiset $A \subseteq V \times V$ of edges, one for each entry point connecting one PA to another PA. The direction of edges reflects the direction of entrypoints. Edges are labelled with the corresponding entrypoints, i.e., $e_{i,j}$ or $e_{i,j}'$, for some $k > 0$ in case of multiple entry points from $pa_i$ to $pa_j$.

If there is a path between the two vertex representing $pa_i$ and $pa_j$ respectively, we say that $pa_j$ is reachable from $pa_i$. If the path consists of a unique edge then $pa_j$ is directly reachable. Main entrance and main exit are the vertices that are directly reachable from $pa_{204}$ and from which $pa_{204}$ can be directly reached, respectively. The graph $G$ is assumed to be connected. Moreover, for the accessibility graph to be consistent with the hierarchy of PA, the following properties must hold.

Properties (Accessibility Graph)

- Every PA must be directly reachable from either the parent or sibling in the hierarchy of PA. This follows from the fact to simplify the model, in our context, PAs can only overlap if one is wholly contained within the other. However, one could extend the model to allow overlapping PAs, at the cost of increased complexity of the hierarchical processing.
that a PA can be only entered if the subject is present in the parent PA.

- The main entrances must correspond to the PAs at the highest level in the hierarchy of PA. This follows from the fact that otherwise a PA could be entered from a PA which is different from the parent or sibling, against the definition of hierarchy of PAs.
- The graph also does not contain loops (i.e., edges do not start from and end to the same vertex).

### 3.2.3 Accessibility graph for a subject

Intuitively, the entrance of a subject into a PA corresponds to the transition from one vertex to an adjacent vertex in the accessibility graph. Key to our definition of the security system is the authentication and confirmation of passage. That is, enforcement of Prox-RBAC requires a physical barrier separating PAs, as well as deployment of a technology that ensures only authorized personnel can pass. Detecting such a movement requires the subject’s presence at an entry point. However, we must also have a way to detect that the subject did, in fact, pass through the entry point, rather than doubling back into the current space. As we will describe in Section 4.4, ensuring the trustworthiness of position is a challenging issue in deployment of spatially aware RBAC systems.

Besides passage confirmation, the other aspect of concern that we consider here regards the authentication of passage. Subjects, in general, are not allowed to enter every PA. We denote with $AuthPA(s)$ the set of PAs that the subject $s$ is authorized to enter. For this set to be consistently defined, every PA in the set must be reachable from the outdoor space and, vice versa, the outdoor space must be reachable from any PA in the set. It can be shown that these requirements are met if the following conditions are satisfied.

**Properties (Authorized PAs)**

- If $pa \in AuthPA(s)$, then the parent of $pa$ must also be in $AuthPA(s)$.
- If $pa \in AuthPA(s)$ and $pa$ is only reachable from one or more siblings in the hierarchy, then all these siblings must also be in $AuthPA(s)$.

The subgraph with vertices that are the authorized PAs and edges that are the subset of edges which connect the authorized PAs is itself an accessibility graph, namely it represents the accessibility graph of the subject. Therefore, because every PA of the set can be reached through the parent or a sibling, when the subject is at an entry point $e_{i,j}$, the system can refer to the accessibility graph of the subject to determine whether the subject is authorized to enter the target PA.

### 3.3 Spatial Roles

As in GEO-RBAC, we augment traditional roles with geographic information. Specifically, a spatial role is the tuple $< r, pa >$, where $r \in R$ and $pa \in P \cup \{ T \}$, where $T$ denotes the reference space. A spatial role $< r, pa >$ is enabled for the subject $s$ if $s$ has activated the traditional role $r$, and, in addition, one of these two conditions is satisfied: a) $s \in pa$; ii) $s \in pa_i$ and $pa_i \subseteq pa$. Note that if subject $s$ has been assigned role $< r, pa >$ with $pa \in P$, then $s$ must be authorized to enter $pa$, i.e., $pa \in AuthPA(s)$, otherwise the role would have never been enabled. The spatial role is then activated if the user wishes to exercise the privileges associated with the role. If user $s$ has activated role $< r, pa >$, we write $< r, pa > \in ActiveRoles(s)$.

While role activation in Prox-RBAC seems intuitive, subtle challenges arise when users are permitted to maintain continuous access across PA boundaries. To be precise, consider a user who moves from $pa_i$ to $pa_j$ and what should happen to $< r, pa_i > \in ActiveRoles(s)$. In order to streamline the user experience, Prox-RBAC keeps the roles active, but updates the location from $pa_i$ to $pa_j$. That is, if $< r, pa_i > \in ActiveRoles(s)$ prior to the transition, then $< r, pa_j > \in ActiveRoles(s)$ after. Additionally, if $s \in pa_i$, then $< r, pa_j > \notin ActiveRoles(s)$. One exception to this automatic activation of $< r, pa_j >$ occurs when mutually exclusive roles are defined. Specifically, if $< r, pa_j >$ would conflict with another activated role $< r', pa_j >$, then both are deactivated in the transition. Although it would be desirable to treat the movement as atomic, we find this insufficient to provide strong location guarantees. For instance, a user could initiate the movement with an entry point, then wait for several seconds before actually passing through the doorway. Thus, our access control policies must account for the time between authorization and the actual passage between the PAs. As such, the transition triggers a movement event, during which the user’s location is identified as $s \in pa_k$, where $pa_k$ is the least common ancestor, the smallest PA that contains both the old and new PAs. That is, $pa_i \subseteq pa_k$, $pa_j \subseteq pa_k$, and either $pa_i \not\subseteq pa_l$ or $pa_j \not\subseteq pa_l$ for any $pa_l \in Children(pa_k)$, where $Children(pa_k)$ is defined by the hierarchy of PAs. In addition, permissions currently in use will be revoked during the movement event only if the permission is not granted for $< r, pa_j >$ and $pa_j \not\subseteq pa_i$. Otherwise, the permissions are retained.

### 3.4 Continuous Proximity Constraints

A proximity constraint is a security requirement that is satisfied by the location of other users. If the constraint must be continuously evaluated for the duration of the user’s access session, then we call the constraint continuous. Proximity constraints are built from three primitive constructs: relative constraint clauses, continuity of usage constraints, and timeouts. Relative constraint clauses define the static presence or absence conditions that must be met. However, mobile environments are inherently dynamic. As such, the latter two constructs are necessary to ensure the relative constraint clause is enforced properly as the environment changes.

A relative constraint clause specifies the proximity requirement of other users in the spatial environment. These clauses can be described as either presence constraints or absence constraints.\(^2\)

\(^2\)The deactivation does not occur if $< r, pa_j >$ would conflict with a role $< r', pa_j >$ that is enabled but not currently activated.
To formulate these conditions, we adopt an intuitive syntax that can be illustrated as follows:

\[ \text{at-most 0 civilian in Room 105} \]

The basic structure consists of an optional cardinality qualifier (e.g., \( \text{at-most} \) or \( \text{at-least} \)), a nonnegative integer specifying the number of subjects, a role (e.g., civilian), and a spatial relationship (e.g., in Room 105). The spatial relationship consists of two parts: a topological relation and a logical location descriptor that identifies a PA. Let \( \mathcal{R}_T \) denote the set of topological relations. In our initial approach, we will only consider a small set of relations, namely \( \mathcal{R}_T = \{ \text{in}, \text{out}, \text{adj} \} \). The location descriptor can be absolute, as was the case here, or it can take the form of \( \text{this}.type \). In this latter structure, the type specifies a level in the hierarchy of PAs, while \( \text{this} \) dictates that the location of the subject fulfilling the role in the type satisfies multiple relative constraints. To allow for such cases, we extend our language with the following rules:

\[ C = C \lor C \quad | \quad C \land C \quad | \quad Q \ \text{a role topo pa} \]

\[ Q = \text{at-most} | \text{at-least} | \epsilon \]

Now, let \( \mathcal{W} \) denote the set of continuity of usage constraints. That is, \( \text{while} (c) \in \mathcal{W} \) and \( \text{when} (c) \in \mathcal{W} \) if \( c \in \mathcal{C} \). Given that timeouts can only apply to \( \text{while} \) constraints, we create a distinguished set \( \mathcal{W}_{\text{while}} \subseteq \mathcal{W} \) that consists exclusively of the constraints \( \text{while} (c) \), where \( c \in \mathcal{C} \). As before, let \( \mathcal{P} \) denote the set of all possible proximity constraints, \( \mathcal{C} \) denotes the set of all possible continuity of usage constraints, and \( \epsilon \) is the absence of a proximity constraint, which allows for traditional spatial awareness policies.

\[ W :: = \mathcal{W} \lor W \quad | \quad \mathcal{W} \land W \quad | \quad \text{when} (C) \quad | \quad \text{while} (C) \]

Finally, let \( \mathcal{T} \) denote the set of timeouts written as \( \text{timeout} t \), where \( t \) denotes a finite unit of time, and let \( \epsilon \in \mathcal{T} \). We write \( \Phi = (\mathcal{W} \star \mathcal{T}) \cup \{ \bot \} \) to denote the set of all possible proximity constraints, where \( \bot \) denotes the absence of a proximity constraint, which allows for traditional spatial awareness policies. For the sake of simplicity, we omit from our grammars any parentheses that can be used to indicate boolean formulas. Given that timeouts can only apply to \( \text{while} \) constraints, we create a distinguished set \( \mathcal{W}_{\text{while}} \subseteq \mathcal{W} \) that consists exclusively of the constraints \( \text{while} (c) \), where \( c \in \mathcal{C} \). As before, let \( \mathcal{P} \) denote the set of all possible proximity constraints, \( \mathcal{C} \) denotes the set of all possible continuity of usage constraints, and \( \epsilon \) is the absence of a proximity constraint, which allows for traditional spatial awareness policies.
in terms of the UCONABC family of core models. Prox-RBAC employs UCONABC semantics, as we require authorizations (A) and conditions (C), but not obligations (B). In all of the semantic specifications below, $\geq$ can denote either the dominance relation on the partially ordered set of roles R or the traditional inequality on integers. The notation $2^S$ refers to the power set of the set $S$.

We start with the simplest case, in which $\mu = \bot$. That is, there is no proximity constraint enforced, and the policy indicates a spatially aware RBAC role as defined in existing works. We can write these semantics formally as a UCON$_{preA_0}$ policy.

```
< role, act, obj, ↓ > - UCON$_{preA_0}$:

- role =< r, pa >; r $\in$ R, pa $\in$ P

- Perms = {o | a | e | o $\in$ O, a $\in$ A}

- ActiveRoles : $S$ $\rightarrow$ $2^R$

- PermittedRoles : Perms $\rightarrow$ $2^R$

- ATTS(S) = {ActiveRoles}

- allowed(s, o, a) $\Rightarrow$ $\exists r$ $\in$ EnabledRoles(pa),
  $s \in$ pa $\land$ r $\in$ ActiveRoles(s) $\land$
  $\exists r'$ $\in$ PermittedRoles(o, a), r $\geq$ r'
```

These semantics state that, if s is allowed to perform o on a, there must be a traditional RBAC role r that is enabled by entering pa, s is physically present there, and s has activated the role. In addition, r must dominate r', which is a traditional RBAC role that is permitted to perform a on o. Obviously, it may be the case that r = r'. However, when hierarchical roles are created, r $\geq$ r' implies that activating r inherits all of the permissions associated with r'. As this is UCON$_{preA_0}$, this policy is checked only once prior to granting access. Finally, note that the same semantics can be applied for traditional RBAC policies, which can be expressed with pa = T, indicating that role activation can occur anywhere. Thus, Prox-RBAC semantics are flexible enough to accommodate more traditional policies.

To define the semantics for when and while constraints, we must define a number of helper functions, which are listed in Figure 3. $TopoSat$ defines the conditions under which topological relations are satisfied. Observe that in and out are not simply satisfied by containment. Instead, Prox-RBAC considers the PA describing the subject’s location at the finest granularity. As an illustration, recall the example in Section 3.2.1 in which Room 204 ⊆ Floor 2; if s $\in$ Room 204, then $TopoSat(s, in, Floor 2)$ returns false. adjy, on the other hand, does not consider the finest granularity of the user’s location. Instead, adjy is satisfied if the subject is in a PA that is immediately reachable from the PA under consideration (or vice versa); furthermore, neither PA can be a parent of the other. $TopoSat$ is used in the context of the Sat function, which determines if a constraint has been satisfied.

Building on these functions, we can define what it means to satisfy a relative constraint clause c $\in$ C. Recall that c can either be a simple clause (c $\in$ C), or it can be a complex clause that is created from disjunctions and/or conjunctions of simple clauses. That is, c =< c$_0, b_1, c_1, ..., b_n, c_n$ >, where c$_0$ is a simple clause and b$_i$ is a Boolean connective. For each simple clause c$_i$, we have c$_i$ =< q, n, role, topa >. First, we apply $\hat{c}_i$ = Cast(c$_i$), which replaces the original pa with the PA of the correct type. Note that Cast only changes c$_i$ if pa is the form this.$\tau$, where $\tau$ indicates a type (e.g., room, suite, floor); that is, Cast converts the relative pa into an absolute pa. Now, we can apply Sat to the individual components of $\hat{c}_i$. This procedure describes the functionality of EvalC, which evaluates a simple constraint clause based on the subject’s location.

Similarly, a when clause $w$ $\in$ W$^*$ consists of a Boolean combination of simple when clauses, each of which contains a Boolean combination of relative constraint clauses. Hence, EvalW evaluates each simple clause independently, then evaluates the resulting Boolean expression. Finally, Eval itself operates on the full when clause, evaluating the Boolean expression that is produced by evaluating each simple when clause. This produces the following UCON$_{preC_0}$ semantics:

```
< role, act, obj, when > - UCON$_{preC_0}$:

- [premises from < role, act, obj, ↓ >]
  Eval : W$^*$ $\times$ P $\rightarrow$ {true, false}
  Location : S $\rightarrow$ P

- preCON = {Eval(when, Location(s))}

- getPreCON(s, o, a) = Eval(when, Location(s))

- allowed(s, o, a) $\Rightarrow$
  preConChecked(getPreCON(s, o, a)) $\land$
  $\exists r$ $\in$ EnabledRoles(pa), s $\in$ pa $\land$ r $\in$ ActiveRoles(s) $\land$
  $\exists r'$ $\in$ PermittedRoles(o, a), r $\geq$ r'
```

```
When constraints with a timeout of 0 (i.e., immediate revocation) are similar to when constraints, with the exception that a subset of the conditions are repeatedly checked as the permission is exercised. To model this behavior, we introduce the WhenToBool function, which takes a Boolean combination of when and while clauses and replaces the when clause with either true or false according to the initial evaluation. The result of this extension is the following UCON$_{preC_0.onC_0}$ semantics:

```
< role, act, obj, while > - UCON$_{preC_0.onC_0}$:

- [premises from < role, act, obj, when >]

- WhenToBool : W$^*$ $\times$ P $\rightarrow$ W$^*$

- preCON = {Eval(when, Location(s))}

- getPreCON(s, o, a) = Eval(when, Location(s))

- allowed(s, o, a) $\Rightarrow$
  preConChecked(getPreCON(s, o, a)) $\land$
  $\exists r$ $\in$ EnabledRoles(pa), s $\in$ pa $\land$ r $\in$ ActiveRoles(s) $\land$
  $\exists r'$ $\in$ PermittedRoles(o, a), r $\geq$ r'

- stopped(s, o, a) $\Rightarrow$
  - preConChecked(getOnCON(s, o, a))
```

The final type of constraint to consider is a while constraint with a timeout t > 0. Here, we introduce X to denote the data structures containing expiration times for permissions. That is, PermExp(s) will return $\chi$ X. The simplest form of x would be a 2-dimensional array $O \times A$, where (o, a) would store the expiration time for exercising a in A on o in O. As such, we use the notation $x[o, a]$ for this value, though we formalize this behavior with the FindExp function. The UpdateExp function takes such a data structure, and updates only the $x[o, a]$ entry to be $t_e + t_e$, the sum of the current system time and the expiration time. All other entries remain unchanged.

Using these functions, we extend the UCON$_{preC_0.onC_0}$ model 5 to express while constraints with a timeout using the following semantics. Our extension is to introduce the onUpdate(ATT(s)) procedure to update the subject’s PermExp attribute as the access occurs. During access, if the condition holds (i.e., the while clause is satisfied), then the expiration is updated accordingly. The permission is revoked only if the condition fails and the current time is greater than the expiration time.

5UCON$_{ABC}$ does not define an onUpdate procedure for conditions, but it does for authorizations and obligations. However, we find it very straightforward to augment the model to support the same functionality for conditions.
4. IMPLEMENTATION

As a proof of concept, we have implemented a Prox-RBAC prototype. Our architectural design couples a centralized authorization server with a distributed, asynchronous clients. These clients can be either stationary (e.g., workstations) or mobile devices (e.g., laptops). In either case, accessing sensitive data requires providing a high-integrity proof of location that is generated with the help of a fixed-location device, such as a near-field communication (NFC) or magnetic stripe reader. In addition, passage through an entry point must be monitored and controlled by a fixed-location device that reports the user’s location change to the centralized server. In this section, we describe the principals of our enforcement architecture, our multi-factor authentication protocol and algorithm for requesting access to a protected resource, and a description of our implementation features. We then use these definitions to present the sketch of a formal proof of correctness.

4.1 Principals

Our Prox-RBAC prototype is built on interaction between four principals. First, the Authorization Server (AS) is the centralized server that acts as the policy decision point (PDP). The AS maintains a mapping of all users’ locations, represented at the finest-granularity PA that applies. The AS also maintains all access control policies. Next, the User refers to the human user requesting access. Each user (subject) s will have an identifier id_s, a password pwd,\(^6\), and a proximity-connection device that is used for generating the proof of location. This portable device will have a secret value sv(s), as well as a certificate, signed by the AS, that contains data required to perform an interactive zero-knowledge proof-of-knowledge protocol that demonstrates possession of sv(s).

The user performs the access control protocol using a Client, which is a trusted computing device for accessing sensitive data. The client acts as the policy enforcement point (PEP), revoking privileges when requested by the AS. As clients may be mobile, they are identified solely by an identifier and denoted c_i. Each client has a network connection to communicate with the AS and is equipped with a trusted computing component (TCC), such as a TPM, that can bind keys to applications. That is, the application requesting access to a protected resource will have a public-private key pair, denoted ph(c_i) and sk(c_i), respectively, and a certificate Cert(c_i) signed by the AS. For simplicity, we assume the presence of a trusted path for the user to enter a password. Additionally, we assume unauthorized software is prevented from accessing sensitive data, and remote attestation techniques are used to ensure that the software on the client matches a pre-approved configuration.

A Location Device (LD) is a fixed-location reader distributed in a PA. These devices are used to authenticate the location of the user at the time of an access request. Since these devices have fixed locations, each is denoted ld_j, to indicate the jth location device in pa_i. Each LD employs a proximity-based communication technology and possesses a certificate Cert(ld_j) that is signed by the AS. These certificates contain coordinates cdlt(ld_j), as well as data that allows AS to retrieve sv(ld_j), a secret value stored on ld_j. For example, this data could be an encrypted version of sv(ld_j), where only AS has the corresponding key. In order to bind the user and client to the location, the client and LD must have a physical connection. For instance, the LD could be integrated in a laptop base station that is built into a desk.\(^7\)

4.2 Access Request

Our access request protocol is built on a number of cryptographic primitives. First, let (Gen, Enc, Dec) denote both public key and symmetric key encryption schemes. In both cases, k ← Gen(1^n)
denotes a probabilistic key generation algorithm and $m := \text{Dec}_k(c)$ denotes the decryption of ciphertext $c$ using the key $k$. In the public key case, $c \leftarrow \text{Enc}_k(m)$ is an IND-CPA-secure encryption (which is probabilistic), while $c := \text{Enc}_k(m)$ denotes $(t, e)$-secure (deterministic) symmetric key encryption. While this reuse is a slight abuse of notation, it should be obvious to the reader which scheme is used, as we denote public key pairs as $pk(\cdot)$ and $sk(\cdot)$ and a symmetric key as $K$.

Similarly, $(\text{Gen}, \text{Sign}, \text{Verify})$ denotes an unforgeable signature scheme where $k \leftarrow \text{Gen}(1^n)$ denotes a probabilistic key generation algorithm, $s := \text{Sign}(m(k))$ indicates the signature $s$ to be the message $m$ signed with the secret key of $x$, and $v := \text{Verify}(pk(a), s, m)$ produces either true or false by verifying $s$ is the signature of $m$, using the public key of $x$. Let $H$ denote a collision-resistant cryptographic hash function such that $h := H(m)$ denotes the hash of a message $m$.

Finally, let $(\text{Commit}, \text{Open})$ denote a non-interactive commitment scheme that is both perfectly hiding and computationally binding. $c \leftarrow \text{Commit}(s)$ refers to the probabilistic generation of the public commitment $c$ for the secret value $s$, while $\text{Open}(c, s)$ denotes the process of revealing of the parameters of the commitment. Let $\text{Prove}(s(p))$ denote an interactive zero-knowledge proof-of-knowledge protocol in which the prover $p$ demonstrates knowledge of a secret value $s$. Finally, $\text{Author}(id_s, pwd_s, r)$ denotes an authentication protocol in which the user provides password $pwd_s$ to prove the claim to identity $id_s$, and $r$ indicates a requested role.

Figure 4 shows our basic access protocol. It is initiated by a subject $s \in S$ to request read permission on object $o \in O$, while using role $r \in R$. The request is made using client $c_l$ at location $ld_{l,j}$. The protocol starts by $s$ using his proximity device to prove knowledge of $sv(s)$ to $ld_{l,j}$ and entering $id_s$, $pwd_s$, $r$ and $r$ into the client via a trusted path. The client signs a version of $id_s$ to $ld_{l,j}$ via the physical connection, and $ld_{l,j}$ responds with a commitment $c$, binding $id_s$ to $c_l$ at timestamp $T$ with a nonce $n$. Note that only $ld_{l,j}$ is able to open this commitment. $c_l$ signs the commitment, sending the result to the AS. The AS and $c_l$ then enter an authentication protocol that confirms the identity of $id_s$ and his authorization to enter $r$.

Assuming the authentication of $s$ is successful, the AS returns a signed version of the commitment $c$, encrypted in a manner that is only readable by $c_l$, $c_s$ decrypts the signed commitment, and forwards the result to $ld_{l,j}$, who confirms the signature of the AS (thus indicating that the AS received the commitment intact). $ld_{l,j}$ opens the commitment and encrypts the result with the public key of the AS. $c_l$ forwards the encrypted packet and the name of the object $o$ requested, and the AS returns the encrypted object with a key bound to $c_l$. Note that, while this protocol describes read actions, it is straightforward to extend it to handle write actions.

### 4.3 Algorithms

Policy evaluation occurs after step 10 of the access control protocol in Figure 4. The Request algorithm, defined below, attempts to activate the spatial role based on the user’s location. If the activation is successful, the policies are evaluated and the on-going conditions (in the case of while constraints) are determined. Assuming the initial conditions are satisfied, the algorithm returns approval and the object is sent in step 11 of the protocol.

**Request($s, o, a, r$)**

Input: $s$: the requesting subject; $o$: the requested object; $c_l$: the client; $a$: the requested action

Output: approved or denied

1. if $\text{Active}(s, r, \text{Location}(s)) = \text{failure}$ then return denied
2. $\text{satisfied} \leftarrow \text{EvalPolicies}(s)$
3. if $\text{satisfied} = \emptyset$ then return denied
4. ongoing \leftarrow ∅
5. for each $p < sr, a, o, r > \in \text{satisfied}$ do
6. ongoing \leftarrow ongoing $\cup$ $\text{EvalToBool}(p, \text{Location}(s))$
7. $\text{Ongoing}(s) = \text{Ongoing}(s) \cup \{s, a, c_l, \text{ongoing} > \}$
8. return approved

The policy evaluation procedure is defined in the EvalPolicies algorithm. The AS identifies the relevant policies for the object and action, determines if the user has an activated role that satisfies the policy, then proceeds to evaluate the proximity constraints. The algorithm returns the set of policies that are satisfied, as these policies may have on-going conditions (i.e., while clauses) that must be continuously enforced.

**EvalPolicies($P(o, a)$)**

Input: $s$: the requesting subject; $P(o, a)$: the set of matching policies

Output: satisfied: the set of policies that have been satisfied

1. foreach $p \in P(o, a)$ do
2. $w \leftarrow \text{in ActiveRoles}(s)$
3. if $\text{w} \geq \text{r}_p$ and $\text{pa}_a \leq \text{w}$ then
4. permitted \leftarrow true
5. if permitted \leftarrow true then
6. $\text{for each} \text{w clause in } \varphi$
7. $\text{for each constraint in } \text{w}$
8. $\text{evaluate constraint based on location information}$
9. $\text{evaluate w clause based on constraint satisfaction}$
10. if $w$ is satisfied then
11. $\text{satisfied} \leftarrow \text{satisfied} \cup \{p\}$
12. return satisfied

The constraint evaluation is a straightforward evaluation based on the location information maintained by AS and the specified topological relation. For instance, consider the constraint

$\text{when at least 1 Supervisor in Room 105}$

The AS simply checks to see if there is at least one person in Room 105 who has Supervisor as an active role. If the constraint is not satisfied

$\text{when at least 1 Supervisor in Floor 1}$

then the AS would compute the sum of all users with an active Supervisor role who are present anywhere on the first floor. If the constraint uses the this keyword, the algorithm traverses the parent hierarchy to find the correct PA based on the user’s PA. Hence, the policy evaluation has a linear complexity based on identifying the PAs that are relevant to the constraint.
4.4 Prototype Implementation

We have developed a proof-of-concept prototype of Prox-RBAC to measure the performance of the cryptographic protocols and the enforcement algorithms. To instantiate the Prove construct, we employed the Feige-Fiat-Shamir identification protocol [11], which uses a zero-knowledge proof, and we use a Pedersen commitment [19] for the Commit and Open primitives. For Auth, we simply used a salted hash of a password. We used SHA-256, AES-256, 1024-bit RSA, and SHA-1 with DSA for the Hash, Enc, EncP(k), and Sign primitives, respectively. We implemented our prototype in Java 1.6.0_20, relying on standard cryptographic implementations when possible. For the Pedersen commitment and the Feige-Fiat-Shamir protocols, we used a custom implementation that employed the BigInteger class. Our test machine consisted of a 2.26GHz Intel® Core™ 2 Duo CPU with 3GB of 667MHz memory, running on Ubuntu 10.04 (“Lucid Lynx”) with version 2.6.32 of the Linux kernel. Based on 500 iterations, the most expensive of the cryptographic operations are the Pedersen commitment (average of 17.7 ms to generate and 20.2 ms to confirm) and RSA (5.7 ms to encrypt, but 30.2 ms to decrypt). Other than the DSA signature (9.8 ms average), all other computations required less than 1 ms on average to complete. The average time for the complete Read protocol (including the policy evaluation algorithms) was approximately 89.4 ms.

Moving toward a practical deployment with location sensing is more challenging. We have performed preliminary work toward using an Advanced Card Systems (ACS) NFC reader, model ACR 122 to communicate with a Nokia 6131 NFC-enabled cell phone. Communication between the ACR 122 and the Nokia 6131 uses the peer-to-peer extension to the Java JSR 257 Contactless Communication API. Our software employs the NFCIP library, which uses the Java smartcardio libraries. One difficulty we had with this implementation is that the BigInteger class does not exist in the Java ME distribution. Consequently, deploying a protocol such as the Feige-Fiat-Shamir scheme requires developing one’s own solution for large integers. However, based on our experiments, we observe that the average computation time for generating the Feige-Fiat-Shamir proof is less than the amount of time to perform the AES encryption in the protocol. Thus, such a deployment is feasible.

The final challenge in developing a practical implementation of Prox-RBAC is enforcement of the one-way movement through entry points. In our current prototype, we use the NFC phone to connect to a reader that would (in a real deployment) control the lock to a door. To confirm passage, the user would have to use the NFC phone to connect to a reader on the other side of the door. Clearly, this approach is inelegant, and a more desirable solution would include other forms of sensing to detect the user’s movement. In addition, our prototype does not include proximity-based communication between the client (i.e., a laptop) and the entry. For completeness and to prevent data leakage, the entry points should be equipped with a means to detect when a laptop is leaving the PA and revoking permissions as necessary.

4.5 Functional Correctness

In this section, we use the protocol and algorithm specifications to demonstrate that our prototype correctly enforces the Prox-RBAC semantics as defined. Due to space constraints, we will only offer sketches of proofs and leave full consideration for the full version of the paper.

Lemma 1: The access protocol defined in Figure 4 prevents unauthorized access under the Dolev-Yao adversarial model.

Proof: As all sensitive data is encrypted, eavesdroppers only observe signed data (preventing unauthorized modification), the object being requested, or the one-time use commitment. If the adversary modifies either of the latter two pieces of data, the user and/or the AS would detect the corruption. Furthermore, modifying the object requested would not give the user access to unauthorized data. Thus, only authorized access is allowed. □

Lemma 2: The policy evaluation algorithms correctly enforce when and while constraints.

Proof: The EvalPolicies algorithm includes a procedure for evaluating the initial conditions, corresponding to the when constraints. The algorithm returns the set of satisfied policies, which are then added to the user’s Ongoing list. This list identifies the while constraints that must continuously be evaluated. If an Ongoing constraint is no longer satisfied, the list entry contains the object and client used; the AS uses this information to send a revocation request to the client. □

5. RELATED WORK

Role-based access control (RBAC) [21] is a popular technique for maintaining and administering access privileges in an organization. One of the advantages of RBAC is its ability to incorporate hierarchies into the model [22, 23]. Building on the core RBAC model, researchers have proposed extensions that integrate contextual factors [6, 8]. These extensions include temporal and spatial constraints [9, 14, 1, 9, 20, 4, 7, 5]. Our work can be viewed as an extension of the latter category, in which spatial information is considered as part of the access control request. However, our work has a significantly different focus than these works.

Specifically, models for spatial RBAC systems, such as those just identified, are based solely on the location of the user making the request. The presence or absence of other users in the reference space is irrelevant in those models. In contrast, our focus is to define proximity constraints that are based on the user’s location in relation to other users in the system. This work is the first to focus on the specification of these relative constraints and the challenges of their enforcement.

Two works are close in aim as our own. Team-based access control using contexts (C-TMAC) [12], like our own, incorporates both RBAC and the location of multiple users. However, in C-TMAC, the system only considers locations of users who are on the same team as the requesting user, rather than all users. Furthermore, C-TMAC does not specifically focus on location, but rather the more generic notion of context (of which location is one aspect). As such, C-TMAC does not consider the challenges of location validation and does not define a policy language. A different interpretation of proximity has previously been explored in the context of access control. Specifically, the proximity-based access control (PBAC) model [10, 13] considers the requesting user’s proximity to a computer. For instance, if a temporary emergency medical center is created during a disaster response, then a doctor who is in the immediate vicinity would be granted access. In other words, PBAC can be viewed as considering absolute proximity, as the computer typically has a static location. Our work, in contrast, considers relative proximity based on the locations of other users, who are assumed to be continually moving.

One area of study that may appear similar to our own is the question of nearest neighbor queries. For instance, Yang et al. [24] have proposed techniques for k-nearest neighbor queries within an indoor environment. These queries are based on a minimal walking distance metric. Although Prox-RBAC also considers the location of other users, we do not use a uniform metric. That is, our spatial
model abstracts the distance between protected regions of varying sizes. One interesting extension of Prox-RBAC could be to apply $k$-nearest neighbor queries to our spatial model, where the distance metric is the number of hops within the network of entry points.

6. CONCLUSIONS

In this work, we have extended the notion of spatially aware RBAC to incorporate proximity constraints, which specify policy requirements that are based on the locations of other users in the environment. We have introduced our spatial model, primarily consisting of an accessibility graph that is based on existing work on graph-based indoor space models. We have also defined the syntax and semantics for the Prox-RBAC language for specifying these constraints. In addition to the formalization of our model and language, we have defined an enforcement architecture, including protocols and algorithms. We have offered preliminary results that prove the architecture correctly meets the semantic definitions. We have also described our initial work toward developing a prototype Prox-RBAC system, and closed with an informal security analysis. Based on these results, we find that it is feasible to construct a usable and efficient proximity-based RBAC system.

7. ACKNOWLEDGEMENTS

The work reported in this paper has been partially supported by Sypris Electronics and by the MURI award FA9550-08-1-0265 from the Air Force Office of Scientific Research.

8. REFERENCES