LRBAC: A Location-Aware Role-Based Access Control Model[*]

Indrakshi Ray, Mahendra Kumar, and Lijun Yu

Department of Computer Science
Colorado State University
Fort Collins, CO 80523-1873
{iray,kumar,lijun}@cs.colostate.edu

Abstract. With the growing use of wireless networks and mobile devices, we are moving towards an era where location information will be necessary for access control. The use of location information can be used for enhancing the security of an application, and it can also be exploited to launch attacks. For critical applications, a formal model for location-based access control is needed that increases the security of the application and ensures that the location information cannot be exploited to cause harm. In this paper, we show how the Role-Based Access Control (RBAC) model can be extended to incorporate the notion of location. We show how the different components in the RBAC model are related with location and how this location information can be used to determine whether a subject has access to a given object. This model is suitable for applications consisting of static and dynamic objects, where location of the subject and object must be considered before granting access.

1 Introduction

With the increase in the growth of wireless networks and sensor and mobile devices, we are moving towards an age of ubiquitous computing where location information will be an important component of access control. For instance, when a user moves away from his computer in a smart room, the access should be automatically denied. When a computer containing top secret information is placed in a public place, the computer should automatically become inaccessible. The traditional access control models, such as Discretionary Access Control (DAC) or Role-Based Access Control (RBAC), cannot provide such location-based access control. These traditional models need to be augmented so that they can provide location-based access.

Denning and MacDoran [4] and other researchers have advocated that location information can be used to provide additional security. For instance, a user should be able to control or fire a missile from specific high security locations only. Moreover, the missile can be fired only when it is in a certain location. For such critical applications, we can include additional checks, such as verification of the location of the user and the location of the missile, that must be satisfied before the user is granted access. With the reduction in cost of Geo-Positional Systems (GPS) and infra-red sensors, this indeed is a viable option.

[*] This material is based upon work funded by AFOSR under Award No. FA9550-04-1-0102

Using location information for providing security has its own drawbacks as well. For example, information about the location of a user can compromise his privacy. Alternately, malicious users can observe the presence of a person in a certain location and infer the activities being performed by the person. The use of location information must be carefully controlled to prevent malicious users from launching attacks. Such attacks may have disastrous consequences for critical applications such as the military. In short, a formal model is needed for performing location-based access control.

In this paper we propose one such formal model that is suitable for commercial applications. Our model is based on RBAC. We show how RBAC can be extended to incorporate the concept of location. We illustrate how the different components in RBAC are related with location and how location impacts these different components. Finally, we show how this location information can be used to determine whether a user has access to a given object. The correct behavior of the model is formulated in terms of constraints that must be satisfied by any application using this model.

The remainder of the paper is organized as follows. Section 2 illustrates how we represent location in our model, and how to protect location information. Section 3 shows how the different components of RBAC are related with location and the constraints that location-based access control imposes on these components. Section 5 mentions some work related to this area. Section 6 concludes the paper with pointers to future directions.

2 Our Approach to Location Formalization

In order to perform location-based access control, we need to perform operations on location information and protect the location information. We begin by formalizing the concept of location. Locations can be specified at different levels of granularity. The smallest granularity of location is a point. A location is formally defined as follows.

Definition 1. [Location] *A location Loc_i is a non-empty set of points $\{p_i, p_j, \ldots, p_n\}$ where a point p_k is represented by three co-ordinates.*

We define three kinds of relations that may exist between a pair of locations. The first one is the *contained in* relation, the second one is the *overlapping* relation, and the third one is the *equality* relation. . The contained in relation formalizes the idea whether one location is enclosed by another. The overlapping relation formalizes the idea whether two locations have one or more points in common. The equality relation determines whether a given pair of locations are the same. These are formally defined below.

Definition 2. [Contained in Relation] *Location Loc_j is said to be contained in Loc_k, denoted as, $Loc_j \subset Loc_k$, if the following condition holds: $\forall p_i \in Loc_j, p_i \in Loc_k$. The location Loc_j is called the contained location and Loc_k is referred to as the containing or the enclosing location.*

Definition 3. [Overlapping Relation] *Location Loc_j is said to overlap Loc_k, denoted as, $Loc_j \bowtie Loc_k$, if the following condition holds: $\exists p_i \in Loc_j$ such that $p_i \in Loc_k$.*

Definition 4. [Equality Relation] *Two locations Loc_i and Loc_j are equal, denoted as $Loc_i = Loc_j$ if $Loc_i \subset Loc_j$ and $Loc_j \subset Loc_i$.*

The contained in relation is reflexive, transitive, and anti-symmetric. The overlapping relation is reflexive, symmetric, and non-transitive. The equality relation is reflexive, symmetric, and transitive. Note that, for any two locations Loc_i and Loc_j, the following holds true: (i) $Loc_i = Loc_j \Rightarrow Loc_i \bowtie Loc_j$ and (ii) $Loc_i \subset Loc_j \Rightarrow Loc_i \bowtie Loc_j$.

We denote the set of all locations as **Loc**. The locations form a partial order where the ordering is described by the contained in relation $\subset$. Since the set of locations **Loc** form a partial order, they can be arranged in the form of a hierarchy. If $Loc_i \subset Loc_j$ and $Loc_i \neq Loc_j$, then Loc_j is higher up in the hierarchy than Loc_i and Loc_j is said to be an ancestor of Loc_i. If $Loc_i \subset Loc_j$ and there is no Loc_k such that $Loc_i \subset Loc_k \subset Loc_j$, then Loc_j is said to be the parent of Loc_i. The root of this hierarchy is occupied by a special location termed "*universe*" that contains every other location.

3 Extending RBAC to Incorporate Location-Based Access Control

Core RBAC embodies the essential aspects of RBAC. The constraints specified by Core RBAC are present in any RBAC application. The model requires that users (human) be assigned to roles (job function), roles be associated with permissions (approval to perform an operation on an object), and users acquire permissions by being members of roles. The model does not place any constraint on the cardinalities of the user-role assignment relation or the permission-role association. Core RBAC also includes the notion of user sessions. A user establishes a session during which he activates a subset of the roles assigned to him. Each user can activate multiple sessions; however, each session is associated with only one user. The operations that a user can perform in a session depend on the roles activated in that session and the permissions associated with those roles.

In this section we show how RBAC can be extended to incorporate location-based access control. The different components of RBAC are *Users*, *Roles*, *Sessions*, *Permissions*, *Objects* and *Operations*. We discuss how each of these components are associated with location. Figure 1 illustrates how these components are related with *Location*. The multiplicity of these relationships are indicated by presence or absence of an arrowhead. The absence of an arrowhead indicates a multiplicity of "one" and the presence of arrowhead indicates a multiplicity of "many". Later in Section 4 we formalize these relationships and list the constraints imposed by our model. The operations in Core RBAC that need modification to provide location-aware access are also described in details Section 4.

Users

We assume that each valid user carries a locating device which is able to track the location of a user. The association between user and location is indicated by the edge labeled *UserLocation* in Figure 1. Each user is associated with one location at any given instant of time. However, a single location may be associated with multiple users. Our formalization of this relationship includes defining a function in Section 4 called *UserLocation* that gives the location associated with a valid user. That is, $UserLocation(u)$ returns the location of user u where u is a valid user.

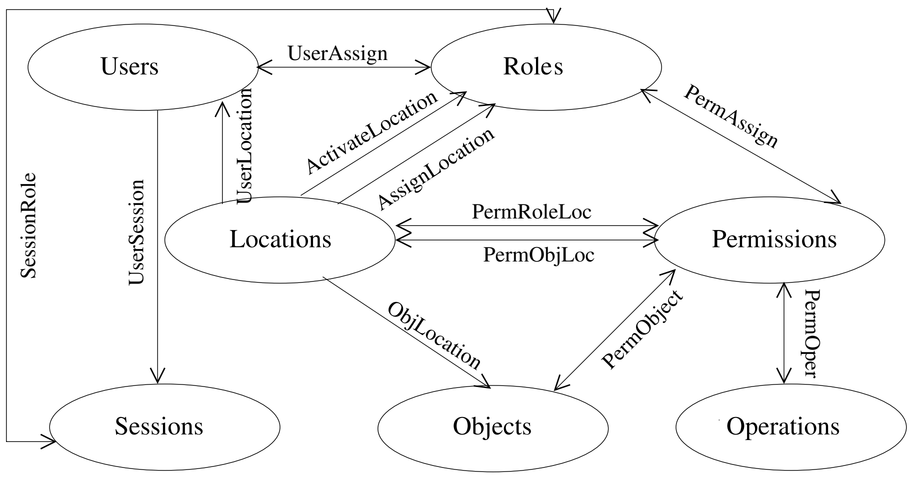

Fig. 1. Relationship of RBAC entities with Location

Roles

In our model, roles are associated with locations. In fact, there are two kinds of associations roles can have with locations. These associations are indicated by the labeled edge *AssignLocation* and *ActivateLocation* in Figure 1. In the following we explain these associations.

Often times, the assignment of user to roles is location dependent. For instance, a person can be assigned the role of U.S. citizen only in certain designated locations. To get the role of conference attendee, a person must register at the conference location. Thus, for a user to be assigned a role, he must be in certain designated locations. In our model, each valid role is associated with a set of locations where the role can be assigned. We define a function *AssignLocation* that takes as input a role and returns the set of locations in which that role can be assigned. In other words, $AssignLocation(r)$ returns the set of locations where a user can be assigned the role r. The assignment of some role s may not be location dependent. In such cases, $AssignLocation(s) = universe$, which implies that the location of the user can be anywhere for it to be assigned the role s. The location of a user u must be contained within $AssignLocation(r)$ for u to be assigned the role r. Since users can be assigned to roles only if they have satisfied the location constraint, the function *AssignUser* in the core RBAC must be changed to include the extra precondition that verifies this.

Some roles can be activated only if the user is in some specific locations. For instance, the role of audience of a theater can be activated only if the user is in the theater. The role of conference attendee can be activated only if the user is in the conference site. Each role is associated with a set of locations from where this role can be activated. The function *ActivateLocation* takes as input a role r and returns the set of locations where the role r can be activated. For a role s whose activation is not location dependent, $ActivateLocation(s)$ equals universe. Before a user u can activate a role r, we must check whether the location of u is contained in $\mathit{ActivateLocation}(r)$. This requires an additional precondition check in the operation *ActivateRole* in the Core RBAC.

Sessions

A user initiates a session. To create a session, a user submits a set of roles that must be activated in that session. If the location of the user satisfies the location constraints for all the roles he is trying to activate, then the session will be created. Otherwise he is notified and the session will not be created. The location associated with the session is the location associated with the user when he created that session. The function *CreateSession* in the Core RBAC needs to be changed because the session creation depends on successful role activation, which, in turn, depends on the location of the user.

Objects

Objects can be physical or logical. Example of physical object is a computer. Files are examples of logical objects. Physical objects have devices that transmit their location information. Logical objects are stored in physical objects. The location of a logical object is the location of the physical object containing the logical object. For the sake of simplicity, we assume that each object is associated with one location only. Each location can be associated with many objects. This is shown by the association *ObjLocation* in Figure 1. The function *ObjLocation* takes as input an object and returns the location associated with the object. Formally, $\mathit{ObjLocation}(o)$ returns the location of object o.

Permissions

Location-based access control models provide more security than their traditional counterparts. This happens because the location of a user and that of an object are taken into account before making the access decisions. Location-based access control also allows us to model real-world requirements where access is contingent upon the location of the user and object. For example, a vendor may provide some free services only to the local customers. Our model should be capable of expressing such requirements.

In the Core RBAC model, permissions are associated with roles, objects and operations. These associations are labeled as *PermAssign*, *PermObject* and *PermOper* respectively in Figure 1. *PermAssign* is a function that takes as input a permission and returns the set of roles assigned to that permission. *PermOper* is a function that takes as input a permission and returns the set of operations associated with this permission. *PermObj* is a function that takes as input a permission and returns the set of objects associated with the permission. A role can perform an operation on an object, if there is a permission that authorizes the role to perform the operation on the object. To incorporate location-aware access, we associate a permission with two locations: allowable role location and allowable object location. These associations are indicated by the edges labeled *PermRoleLoc* and *PermObjLoc* in Figure 1. The function *PermRoleLoc* takes as input a permission p and returns the set of allowable locations for the role associated with this permission. The function *PermObjLoc* takes as input a permission p and returns the set of allowable locations for the object.

In location based access control, a user u is allowed to perform an operation op on an object o in a session s, if there is a permission p such that $\mathit{PermAssign}(p)$ includes an activated role r, $\mathit{PermOper}(p)$ includes op, $\mathit{PermObject}(p)$ includes o, the $\mathit{UserLocation}(u)$ is contained in $\mathit{PermRoleLoc}(p)$ and the $\mathit{ObjLocation}(o)$ is contained in $\mathit{PermObjLoc}(p)$. This requires changing the operation *CheckAccess* in Core RBAC.

$\mathbb{N}$	Set of Natural Numbers
$\mathbb{P}A$	Powerset of Set A
$\setminus$	Set Difference (Also schema 'hiding')
$x \mapsto y$	Ordered Pair (x, y)
$A \rightarrow\!\!\!\!\!+ B$	Partial Function from A to B
$B \lhd A$	Relation A with Set B Removed from Domain
$A \rhd B$	Relation A with Range Restricted to Set B
$\mathrm{dom} A$	Domain of Relation A
$\mathrm{ran} A$	Range of Relation A
$A \oplus B$	Function A Overridden with Function B
$x?$	Variable $x?$ is an Input
$x!$	Variable $x!$ is an Output
x	State Variable x before an Operation
x'	State Variable x' after an Operation
ΔA	Before and After State of Schema A
ΞA	ΔA with No Change to State

Table 1. Relevant Z Notation

$[USER, ROLE, SESSION, LOCATION, OBJECT, OPERATION, PERMISSION]$
$BOOLEAN ::= True \mid False$

4 A Formal Model

In this section, we propose a formal model that extends the core RBAC with location constraints. We use the Z specification language [7] for presenting our formal model. Z is based on set theory, first order predicate logic, and a schema calculus to organize large specifications. Table 1 briefly explains the Z notations used in our examples. Other specification and analysis conventions peculiar to Z are explained as the need arises.

The specification assumes the given types *USER*, *ROLE*, *SESSION*, *PERMISSION*, *LOCATION*, *OBJECT*, *OPERATION*, which enumerate all possible users, roles, sessions, permissions, locations, objects and operations respectively. In addition, we have the enumerated type BOOLEAN that can take on any two values *True* or *False*.

In Z, states, as well as operations, are described with a two-dimensional notation called a *schema*. The declarations for the various objects appear in the top part and the constraints on these objects appear at the bottom part. The objects in the location-aware RBAC are listed in the schema *LRBAC*, which defines the state of the access control system. The objects *Users*, *Roles*, *Sessions*, *Permissions*, *Objects*, *Operations* and *Locations* store the set of users, roles, sessions, permissions, objects, operations, and locations, respectively, of the system. The object *UserLocation* and *ObjLocation* are functions that record the location of the user and the location of the object respectively. The function *AssignLocation* maps a role to a set of locations – these are the locations in which the role can be assigned to some user. *ActivateLocation* is a function mapping a role to the set of locations from where the role can be activated. *UserSession* is a function that returns the set of sessions associated with any user. The function *UserAssign* maps a user to the set of roles assigned to the user. *SessionRole* maps a session to the set of roles that are activated in a session. The function *SessionUser* maps a session to the user associated with that session. The functions *PermObject*, *PermOper*, and *PermAssign* give the set of objects, operations, and roles, respectively, associated with a given permission. The functions *PermRoleLoc* and *PermObjLoc* give the set of allowable locations for the role and object respectively associated with a given permission.

LRBAC

$$
\begin{array}{l}
Users : \mathbb{P}(USER);\ Roles : \mathbb{P}(ROLE);\ Sessions : \mathbb{P}(SESSION);\ Objects : \mathbb{P}(OBJECT) \\
Permissions : \mathbb{P}(PERMISSION) \\
Operations : \mathbb{P}(OPERATION);\ Locations : \mathbb{P}(LOCATION) \\
UserLocation : USER \nrightarrow LOCATION;\ ObjLocation : OBJECT \nrightarrow LOCATION \\
AssignLocation : ROLE \nrightarrow \mathbb{P}(LOCATION);\ ActivateLocation : ROLE \nrightarrow \mathbb{P}(LOCATION) \\
UserSession : USER \nrightarrow \mathbb{P}(SESSION);\ UserAssign : USER \nrightarrow \mathbb{P}(ROLE) \\
SessionRole : SESSION \nrightarrow \mathbb{P}(ROLE);\ SessionUser : SESSION \nrightarrow USER \\
PermObject : PERMISSION \nrightarrow \mathbb{P}(OBJECT);\ PermOper : PERMISSION \nrightarrow \mathbb{P}(OPERATION) \\
PermAssign : PERMISSION \nrightarrow \mathbb{P}(ROLE);\ PermRoleLoc : PERMISSION \nrightarrow \mathbb{P}(LOCATION) \\
PermObjLoc : PERMISSION \nrightarrow \mathbb{P}(LOCATION) \\
\hline
\mathrm{dom}\, UserLocation = Users;\ \mathrm{dom}\, ObjLocation = Objects;\ \mathrm{ran}\, ObjLocation \subseteq Locations \\
\mathrm{dom} AssignLocation = Roles;\ \forall\, l : \mathrm{ran} AssignLocation \bullet l \subseteq Locations \\
\mathrm{dom} ActivateLocation = Roles;\ \forall\, l : \mathrm{ran} ActivateLocation \bullet l \subseteq Locations \\
\mathrm{dom}\, UserSession \subseteq Users;\ \forall\, l : \mathrm{ran}\, UserSession \bullet l \subseteq Sessions \\
\mathrm{dom}\, UserAssign \subseteq Users;\ \forall\, r : \mathrm{ran}\, UserAssign \bullet r \subseteq Roles \\
\mathrm{dom} SessionRole = Sessions;\ \forall\, r : \mathrm{ran} SessionRoles \bullet r \subseteq Roles \\
\forall\, s : Sessions \bullet (SessionRole(s) \subseteq UserAssign(SessionUser(s))) \\
\mathrm{dom} SessionUser = Sessions;\ \mathrm{ran} SessionUser \subseteq Users;\ \mathrm{dom}\, UserSession = \mathrm{ran} SessionUser \\
\mathrm{dom} PermObject = Permissions;\ \forall\, r : \mathrm{ran} PermObject \bullet r \subseteq Objects \\
\mathrm{dom} PermOper = Permissions;\ \forall\, r : \mathrm{ran} PermOper \bullet r \subseteq Operations \\
\mathrm{dom} PermAssign = Permissions;\ \forall\, r : \mathrm{ran} PermAssign \bullet r \subseteq Roles \\
\mathrm{dom} PermRoleLoc = Permissions;\ \forall\, r : \mathrm{ran} PermRoleLoc \bullet r \subseteq Locations \\
\mathrm{dom} PermObjLoc = Permissions;\ \forall\, r : \mathrm{ran} PermObjLoc \bullet r \subseteq Locations
\end{array}
$$

The bottom of the schema *LRBAC* gives the constraints that are imposed on the objects. The first one states that the domain of *UserLocation* equals the set of users in the system. In other words, all users are associated with a location. The next one states that the domain of *ObjLocation* equals the set of objects in the system – this is because all objects are associated with locations. Each location associated with an object must be in an allowable location. This is imposed by requiring that the set of locations in the range of function *ObjLocation* is a subset of *Locations*. The next one states that the domain of *AssignLocation* equals the set of *Roles*. This implies that each role in *Roles* is associated with a set of locations where the role can be assigned. Each set in the range of *AssignLocation* must be a subset of locations of interest in the system.

Similar constraints are imposed for the function *ActivateLocation*. Since all users in the set *Users* may not have an active session, the domain of *UserSession* is a subset of the set *Users*. Each set in the range of *UserSession* denotes the sessions associated with a particular user. Each such set must be a subset of *Sessions*. The function *UserAssign* maps a user to the set of roles assigned to him. Not all users in *Users* may have roles assigned to them, hence the domain of *UserAssign* is a subset of *Users*. Each set in the range *UserAssign* corresponds to roles associated with a user – each such set is a subset of *Roles*. The function *SessionRole* maps a session to a set of roles associated with the session. Each session must be associated with some roles; the domain of *SessionRole* therefore equals *Sessions*. Moreover, each set in the range of *SessionRole* is a subset of *Roles*. The domain of *SessionUser* must equal *Sessions* because every session in *Sessions* must be associated with a user. The range of *SessionUser* is a subset of *Users* because not all users in the system may have an active session. Also, the domain of *UserSession* must equal the range of *SessionUser*. For all sessions, the roles associated with the session must be a subset of the roles assigned to the user associated with that session. Each permission must be associated with a set of objects; thus the domain of *PermObject* equals *Permissions*, and each set in the range of *PermObject* is a subset of *Objects*. Similar constraints are placed for *PermOper*, *PermAssign*, *PermRoleLoc*, and *PermObjLocation*.

$$\begin{array}{|l}
\textit{AddRoleAssignLocation} \\
\hline
\Delta LRBAC;\ r? : ROLE;\ l? : \mathbb{P}\,LOCATION \\
\hline
r? \in Roles;\ l? \subseteq Locations;\ Operations' = Operations;\ Permissions' = Permissions \\
Roles' = Roles;\ Users' = Users;\ ActivateLocation' = ActivateLocation \\
AssignLocation' = AssignLocation \oplus \{r \mapsto AssignLocation(r?) \cup l?\} \\
Locations' = Locations;\ UserLocation' = UserLocation;\ UserSession' = UserSession \\
PermAssign' = PermAssign;\ SessionRole' = SessionRole;\ SessionUser' = SessionUser \\
PermObject' = PermObject;\ PermOper' = PermOper;\ PermRoleLoc' = PermRoleLoc \\
ObjLocation' = ObjLocation;\ PermObjLoc' = PermObjLoc \\
UserAssign' = UserAssign;\ Sessions' = Sessions;\ Objects' = Objects \\
\hline
\end{array}$$

We next describe the operations for performing the administration commands for the Location Aware RBAC. For lack of space, we do not specify all the operations. We show only those operations that are significantly different from the Core RBAC and those that are not present in the Core RBAC. The operations *AddUser*, *DeleteUser*, *AddRole*, *DeleteRole* are similar to that specified in CoreRBAC. Since roles are associated with an assignment location and an activation location, we must have operations that add and delete these associations. The operation *AddRoleAssignLocation* associates a new set of locations with a role. The operation takes as input a role $r?$ and a set of locations $l?$. It has two preconditions. The first verifies whether $r?$ is a valid role by checking whether $r?$ is in the set *Roles*. The second checks whether $l?$ is a subset of *Locations*. The post condition changes the object *AssignLocation*. The role $r?$ is now mapped to the set of original locations union $l?$. All the other objects remain unchanged.

The operation *DeleteRoleAssignLocation* also takes as input a role $r?$ and a set of locations $l?$. This has two preconditions. The first one verifies role $r?$ is a valid role by checking whether $r?$ is in the set *Roles*. The second one checks whether $l?$ is a subset of the locations that are associated with the assignment of role $r?$. The postcondition removes $l?$ from the set of roles associated with $r?$. No other objects are changed. The operations *AddRoleActivateLoc* and *DeleteRoleActivateLoc* are specified in a similar fashion.

DeleteRoleAssignLocation

$\Delta LRBAC;\ r? : ROLE;\ l? : \mathbb{P}\, LOCATION$

$r? \in Roles;\ l? \subseteq AssignLocation(r?);\ Users' = Users;\ Roles' = Roles$
$AssignLocation' = AssignLocation \oplus \{r \mapsto AssignLocation(r?) \setminus l?\}$
$Permissions' = Permissions;\ Sessions' = Sessions;\ Objects' = Objects$
$Operations' = Operations;\ Locations' = Locations;\ UserLocation' = UserLocation$
$UserSession' = UserSession;\ SessionRole' = SessionRole;\ PermAssign' = PermAssign$
$ActivateLocation' = ActivateLocation;\ SessionUser' = SessionUser;\ PermOper' = PermOper$
$PermObject' = PermObject;\ PermRoleLoc' = PermRoleLoc;\ ObjLocation' = ObjLocation$
$PermObjLoc' = PermObjLoc;\ UserAssign' = UserAssign$

AddRoleActivateLoc

$\Delta LRBAC;\ r? : ROLE;\ l? : \mathbb{P}\, LOCATION$

$r? \in Roles;\ l? \subseteq Locations;\ PermObjLoc' = PermObjLoc;\ Users' = Users$
$ActivateLocation' = ActivateLocation \oplus \{r \mapsto ActivateLocation(r?) \cup l?\};\ Objects' = Objects$
$Operations' = Operations;\ Locations' = Locations;\ UserLocation' = UserLocation$
$UserSession' = UserSession;\ PermAssign' = PermAssign;\ Sessions' = Sessions$
$Roles' = Roles;\ AssignLocation' = AssignLocation;\ ObjLocation' = ObjLocation$
$Permissions' = Permissions;\ SessionRole' = SessionRole;\ SessionUser' = SessionUser$
$PermObject' = PermObject;\ PermOper' = PermOper;\ PermRoleLoc' = PermRoleLoc$

DeleteRoleActivateLoc

$\Delta LRBAC;\ r? : ROLE;\ l? : \mathbb{P}\, LOCATION$

$r? \in Roles;\ l? \subseteq AssignLocation(r?);\ Users' = Users;\ Operations' = Operations$
$AssignLocation' = AssignLocation \oplus \{r \mapsto AssignLocation(r?) \setminus l?\}$
$Objects' = Objects;\ Locations' = Locations;\ UserLocation' = UserLocation$
$Sessions' = Sessions;\ UserSession' = UserSession;\ PermAssign' = PermAssign$
$ActivateLocation' = ActivateLocation;\ ObjLocation' = ObjLocation;\ PermOper' = PermOper$
$SessionRole' = SessionRole;\ SessionUser' = SessionUser;\ PermObject' = PermObject$
$PermRoleLoc' = PermRoleLoc;\ Roles' = Roles;\ PermObjLoc' = PermObjLoc$

$$\begin{array}{|l}
\multicolumn{1}{l}{\underline{\mathit{AddPermission}}}\\
\Delta \mathit{LRBAC};\ p? : \mathit{PERMISSION};\ obj? : \mathbb{P}\,\mathit{OBJECT}\\
r? : \mathbb{P}\,\mathit{ROLE};\ op? : \mathbb{P}\,\mathit{OPERATION};\ rl?, ol? : \mathbb{P}\,\mathit{LOCATION}\\
\hline
p? \notin \mathit{Permissions};\ obj? \subseteq \mathit{Objects};\ op? \subseteq \mathit{Operations};\ rl? \subseteq \mathit{Locations}\\
\mathit{AssignLocation}' = \mathit{AssignLocation};\ ol? \subseteq \mathit{Locations};\ \mathit{ActivateLocation}' = \mathit{ActivateLocation}\\
\mathit{Users}' = \mathit{Users};\ \mathit{Permissions}' = \mathit{Permissions} \cup \{p?\};\ \mathit{Sessions}' = \mathit{Sessions}\\
\mathit{PermAssign}' = \mathit{PermAssign} \cup \{p? \mapsto r?\};\ \mathit{PermObject}' = \mathit{PermAssign} \cup \{p? \mapsto obj?\}\\
\mathit{PermOper}' = \mathit{PermAssign} \cup \{p? \mapsto op?\};\ \mathit{PermRoleLoc}' = \mathit{PermRoleLoc} \cup \{p? \mapsto rl?\}\\
\mathit{PermObjLoc}' = \mathit{PermObjLoc} \cup \{p? \mapsto ol?\};\ \mathit{Objects}' = \mathit{Objects};\ \mathit{Roles}' = \mathit{Roles}\\
\mathit{ObjLocation}' = \mathit{ObjLocation};\ \mathit{Operations}' = \mathit{Operations}\\
\mathit{Locations}' = \mathit{Locations};\ \mathit{UserLocation}' = \mathit{UserLocation};\ \mathit{UserSession}' = \mathit{UserSession}\\
\mathit{UserAssign}' = \mathit{UserAssign};\ \mathit{SessionRole}' = \mathit{SessionRole};\ \mathit{SessionUser}' = \mathit{SessionUser}\\
\hline
\end{array}$$

$$\begin{array}{|l}
\multicolumn{1}{l}{\underline{\mathit{DeletePermission}}}\\
\Delta \mathit{LRBAC};\ p? : \mathit{PERMISSION}\\
\hline
p? \in \mathit{Permissions};\ \mathit{Users}' = \mathit{Users};\ \mathit{Roles}' = \mathit{Roles};\ \mathit{Sessions}' = \mathit{Sessions}\\
\mathit{Permissions}' = \mathit{Permissions} - \{p?\};\ \mathit{PermAssign}' = \{p?\} \lhd \mathit{PermAssign}\\
\mathit{PermObject}' = \{p?\} \lhd \mathit{PermObject};\ \mathit{PermOper}' = \{p?\} \lhd \mathit{PermOper}\\
\mathit{PermRoleLoc}' = \{p?\} \lhd \mathit{PermRoleLoc};\ \mathit{PermObjLoc}' = \{p?\} \lhd \mathit{PermObjLoc}\\
\mathit{Operations}' = \mathit{Operations};\ \mathit{UserLocation}' = \mathit{UserLocation};\ \mathit{UserAssign}' = \mathit{UserAssign}\\
\mathit{Locations}' = \mathit{Locations};\ \mathit{Objects}' = \mathit{Objects};\ \mathit{AssignLocation}' = \mathit{AssignLocation}\\
\mathit{ActivateLocation}' = \mathit{ActivateLocation};\ \mathit{UserSession}' = \mathit{UserSession}\\
\mathit{ObjLocation}' = \mathit{ObjLocation};\ \mathit{SessionRole}' = \mathit{SessionRole};\ \mathit{SessionUser}' = \mathit{SessionUser}\\
\hline
\end{array}$$

The operation *AddPermission* adds a new permission to the set *Permissions*. It takes as input the new permission $p?$, the set of roles $r?$ which must be assigned this permission, the set of objects $obj?$ and the set of operations $op?$ associated with this permission, the set of role locations $rl?$ and the set of object locations $ol?$ associated with this permission. There are five preconditions associated with this operation. The first one checks that $p?$ is not an existing permission, the second and third check whether $obj?$ and $op?$ are valid sets of objects and operations respectively, and the fourth and fifth check whether $rl?$ and $ol?$ are valid sets of locations. The postconditions change the set *Permissions* to include $p?$, change the function *PermAssign* to include the new mapping $p? \mapsto r?$. The function *PermObject*, *PermOper*, *PermRoleLoc*, and *PermObjLoc* are updated to include the mappings of permission $p?$ to object $obj?$, operation $op?$, role location $rl?$, and object location $ol?$ respectively. The other objects remain unchanged.

The operation *DeletePermission* removes a permission $p?$ from the set *Permissions*. It takes in one input: the permission $p?$ to be removed. The precondition checks whether $p?$ is an existing permission by checking if it is in the set *Permissions*. The postcondition removes $p?$ from *Permissions*. It also changes the objects *PermAssign*, *PermObject*, *PermOper*, *PermRoleLoc*, *PermObjLoc* by removing the entries associated with $p?$ in each of these objects.

The operation *AssignUser* must also be changed. It takes in two inputs: the user $u?$ to whom the role $r?$ must be assigned. This operation has four preconditions. The first two check whether $u?$ and $r?$ are valid user and role respectively. The third one ensures that $r?$ is not in the set of roles assigned to $u?$. Since $r?$ can be assigned only in certain locations, the last precondition checks whether the location of the user is contained in the locations where $r?$ can be assigned. The postcondition is that the function *UserAssign* is changed so that $u?$ maps to the set consisting of the roles previously assigned to $u?$ and $r?$. The other objects remain unchanged.

AssignUser

$$\begin{array}{l}
\Delta LRBAC;\ u? : USER;\ r? : ROLE \\
\hline
u? \in Users;\ r? \in Roles;\ r? \notin UserAssign(u?);\ UserLocation(u?) \in AssignLocation(r?) \\
PermObjLoc' = PermObjLoc;\ Roles' = Roles;\ Objects' = Objects;\ Operations' = Operations \\
UserAssign = UserAssign \oplus \{u? \mapsto UserAssign(u?) \cup \{r?\}\};\ PermRoleLoc' = PermRoleLoc \\
Locations' = Locations;\ Permissions' = Permissions;\ UserLocation' = UserLocation \\
UserSession' = UserSession;\ PermAssign' = PermAssign;\ ObjLocation' = ObjLocation \\
ActivateLocation' = ActivateLocation;\ SessionRole' = SessionRole;\ SessionUser' = SessionUser \\
PermObject' = PermObject;\ PermOper' = PermOper;\ Users' = Users;\ Sessions' = Sessions
\end{array}$$

AddLocation

$$\begin{array}{l}
\Delta LRBAC;\ l? : LOCATION \\
\hline
l? \notin Locations;\ Users' = Users;\ Roles' = Roles;\ Locations' = Locations \cup \{l?\} \\
Objects' = Objects;\ Operations' = Operations;\ UserLocation' = UserLocation \\
UserSession' = UserSession;\ PermAssign' = PermAssign;\ ActivateLocation' = ActivateLocation \\
Sessions' = Sessions;\ SessionRole' = SessionRole;\ SessionUser' = SessionUser \\
PermObject' = PermObject;\ PermOper' = PermOper;\ PermRoleLoc' = PermRoleLoc \\
ObjLocation' = ObjLocation;\ PermObjLoc' = PermObjLoc;\ Permissions' = Permissions
\end{array}$$

DeleteLocation

$$\begin{array}{l}
\Delta LRBAC;\ l? : LOCATION \\
\hline
l? \in Locations;\ l? \notin \operatorname{ran} ObjLocation;\ l? \notin \operatorname{ran} PermRoleLocl? \notin \operatorname{ran} PermObjLoc \\
l? \notin \operatorname{ran} AssignLocation;\ l? \notin \operatorname{ran} ActivateLocation;\ Locations' = Locations \setminus \{l?\} \\
Operations' = Operations;\ UserLocation' = UserLocation;\ UserSession' = UserSession \\
PermAssign' = PermAssign;\ ActivateLocation' = ActivateLocation;\ ObjLocation' = ObjLocation \\
SessionRole' = SessionRole;\ SessionUser' = SessionUser;\ PermObject' = PermObject \\
PermOper' = PermOper;\ Permissions' = Permissions;\ Roles' = Roles;\ PermRoleLoc' = PermRoleLoc \\
Users' = Users;\ PermObjLoc' = PermObjLoc;\ Sessions' = Sessions;\ Objects' = Objects
\end{array}$$

Two operations, not present in the Core RBAC, are added to this model. The first is *AddLocation* which adds a new location to the set *Locations*. The precondition is that the input $l?$ is not among the existing set *Locations*. The postcondition is that the set *Locations* is changed to include $l?$. The rest of the objects remain unchanged. The second operation is *DeleteLocation*. It removes a location $l?$ from the set *Locations*. The first precondition is that $l?$ must be an existing location. The second one checks that $l?$ must not be the location of any object. The next four preconditions check that $l?$ must not be a location which is associated with some permission, role assignment or role activation. The postcondition removes $l?$ from the set *Locations*. Other objects remain unchanged.

CreateSession

$$\Delta LRBAC;\ u? : USER;\ s? : SESSION;\ rset? : \mathbb{P}(ROLE)$$

$$\begin{array}{l}
u? \in Users;\ rset? \subseteq Roles;\ s? \notin Sessions;\ rset? \in UserAssign(u?) \\
\forall\, r \in rset? \bullet (UserLocation(u) \in ActivateLocation(r));\ Sessions' = Sessions \cup \{s?\} \\
Users' = Users;\ UserSession' = UserSession \oplus \{u? \mapsto UserSession(u?) \cup \{s?\}\} \\
SessionRole' = SessionRole \cup \{s? \mapsto rset?\};\ SessionUser' = SessionUser \cup \{s? \mapsto u?\} \\
AssignLocation' = AssignLocation;\ Objects' = Objects;\ Operations' = Operations \\
Permissions' = Permissions;\ PermAssign' = PermAssign;\ ActivateLocation' = ActivateLocation \\
Roles' = Roles;\ PermObject' = PermObject;\ UserLocation' = UserLocation \\
PermOper' = PermOper;\ UserAssign' = UserAssign;\ PermRoleLoc' = PermRoleLoc \\
PermObjLoc' = PermObjLoc;\ Sessions' = Sessions;\ ObjLocation' = ObjLocation
\end{array}$$

ActivateRole

$$\Delta LRBAC;\ u? : USER;\ r? : ROLE;\ s? : SESSION$$

$$\begin{array}{l}
u? \in Users;\ r? \in Roles;\ s? \in Sessions;\ r? \in UserAssign(u?);\ s? \in UserSession(u?) \\
UserLocation(u?) \in ActivateLocation(r?);\ Users' = Users;\ Objects' = Objects \\
Sessions' = Sessions;\ SessionRole' = SessionRole \oplus \{s? \mapsto SessionRole(s?) \cup \{r?\}\} \\
Operations' = Operations;\ UserLocation' = UserLocation;\ UserSession' = UserSession \\
Roles' = Roles;\ PermAssign' = PermAssign;\ ActivateLocation' = ActivateLocation \\
SessionRole' = SessionRole;\ SessionUser' = SessionUser;\ PermObject' = PermObject \\
PermOper' = PermOper;\ UserAssign' = UserAssign;\ PermRoleLoc' = PermRoleLoc \\
PermObjLoc' = PermObjLoc;\ ObjLocation' = ObjLocation;\ Permissions' = Permissions
\end{array}$$

CheckAccess

$$\Xi LRBAC;\ s? : SESSION;\ op? : OPERATION;\ o? : OBJECT;\ res! : BOOLEAN$$

$$\begin{array}{l}
s? \in Sessions;\ o? \in Objects;\ op? \in Operations \\
\exists\, r : ROLE \bullet (r \in SessionRole(s?) \wedge (\exists\, p : PERMISSION \bullet (r \in PermAssign(p) \wedge op? \in PermOper(p) \\
\qquad \wedge\ o? \in PermObject(p) \wedge ObjLocation(o?) \in PermObjLoc(p) \\
\qquad \wedge\ UserLocation(SessionUser(s?)) \in PermRoleLoc(p)) \Rightarrow res! = TRUE \\
\nexists\, r : ROLE \bullet (r \in SessionRole(s?) \wedge (\exists\, p : PERMISSION \bullet (r \in PermAssign(p) \wedge op? \in PermOper(p) \\
\qquad \wedge\ o? \in PermObject(p) \wedge ObjLocation(o?) \in PermObjLoc(p) \\
\qquad \wedge\ UserLocation(SessionUser(s?)) \in PermRoleLoc(p))) \Rightarrow res! = FALSE
\end{array}$$

Next, we specify the operations for system functions for the Location Aware RBAC. Here again, we specify only those operations that differ from the Core RBAC. The first one is the *CreateSession* operation. This operation takes as input a user $u?$ whose session is being created, the session $s?$ that must be created, and a set of roles $rset?$ that must be activated. This operation has three preconditions. The first ensures that $u?$ is a valid user, that is, it must belong to the set *Users*. The second checks whether $rset?$ is a subset of *Roles*. The third ensures that $s?$ is not an existing session. The fourth precondition checks that $rset?$ is a subset of the roles assigned to the user. The fifth checks that for each role r in the set $rset?$, the location of user $u?$ is contained in the activation location of r. The post condition changes *Sessions* by including $s?$. The function *UserSession* is changed – user $u?$ is now associated with the set of pre-existing sessions of $u?$ together with the session $s?$. The function *SessionRole* is also changed; a new mapping $s? \mapsto rset?$ is added to *SessionRole*. Similarly *SessionUser* is changed to include the mapping $s? \mapsto u?$. All other objects remain unchanged.

The other operation that is significantly different is *ActivateRole*. The input to this operation are the user $u?$, the role $r?$, and the session $s?$ in which the role must be activated. The first three preconditions check whether $u?$, $r?$, and $s?$ are existing user, role and session respectively. The fourth and fifth preconditions check whether $r?$ is a role assigned to the user $u?$, and $s?$ is a session initiated by user $u?$. The last precondition checks that the location of $u?$ is within the activation location of $r?$. The postcondition is that the function *SessionRole* is updated; $s?$ is now associated with $r?$ and also the roles that were previously activated. The other operations remain unchanged.

The next operation is the *CheckAccess* operation. This function takes in three inputs: the session $s?$, the operation $op?$ and the object $o?$. The operation has three preconditions that check whether $s?$, $op?$ and $o?$ are valid session, operation, and object, respectively. The postcondition does not change any object. This operation produces an output $res!$ which is BOOLEAN. If there is a role r that is activated in the session $s?$, and there is a permission p such r, $op?$ and $o?$ are associated with permission p, and the location of the user and the object are contained in $PermRoleLoc(p)$ and $PermObjLoc(p)$, respectively, then $res!$ returns true. Otherwise $res!$ returns false.

5 Related Work

Denning and MacDoran [4] propose many motivating examples as to why location-based security is important for applications. Their argument is that use of location information can enhance the security of applications. The authors discuss how location-based authentication can be achieved using GPS and a tool called Cyberlocator.

Hengartner and Steenkiste [5] design an access control mechanism for a people location system. The authors say that an individual is associated with two kinds of location policies. The first are the location policies that are specified by individuals. The second are the policies that are specified by institutions. For instance, if a person is at the mall, his individual location policies are in effect. When he is at work, the institutional policies of the organization are in effect. The access control policy is enforced using digital certificates.

Leonhardt and Magee [6] discuss how location-based access can be provided over existing matrix-based access control models and mandatory access control models. The proposed approach consists of three parts: controlling access, controlling visibility and controlling anonymity. The visibility policy specifies what location information gets returned to the user who is querying about the location of an entity. The anonymity policy specifies what information gets returned to the user querying about some other user. The access policy specifies how location information can be used to control access. Although this paper has presented some nice ideas, it lacks details and formalisms. For instance, the authors do not discuss how the different components of an access control model are impacted by location and what constraints are necessary on the location-based model. In our work we try to address these issues and complement the above mentioned work.

Sampemane et al. [9] present a new access control model for active spaces. Active space denotes the computing environment integrating physical spaces and embedded computing software and hardware entities. The active space allows interactive exchange of information between the user and the space. Environmental aspects are adopted into the access control model for active spaces, and the space roles are introduced into the implementation of the access control model based on RBAC. The model supports specification of RBAC policies in which system administrator maintains the access matrix and DAC policies in which users create and update security policies for their devices.

Covington et al. [3] introduce environment roles in a generalized RBAC model (GR-BAC) to help control access control to private information and resources in ubiquitous computing applications. The environments roles differ from the subject roles in RBAC but do have similar properties including role activation, role hierarchy and separation of duty. In the access control framework enabled by environment roles, each element of permission assignment is associated with a set of environment roles, and environment roles are activated according to the changing conditions specified in environmental conditions, thus environmental properties like time and location are introduced to the access control framework. In a subsequent work [2], Covington et al. describes the Context-Aware Security Architecture (CASA) which is an implementation of the GR-BAC model. The access control is provided by the security services in the architecture. In CASA, polices are expressed as roles and managed by the security management service, authentication and authorization services are used to verify user credentials and determine access to the system resources. The environmental role activation services manage environmental role activation and deactivation according to the environment variables collected by the context management services.

Ray et al.[8] briefly describe the impact that location has on the various components of RBAC. However, the paper does not provide any formal treatment of the topic. Bertino et al.[1] describe the GEO-RBAC model which extends RBAC to incorporate spatial and location information. Each role is associated with an extent which defines the boundary at which the role is effective. Our model not only associates locations with role activations, but also associates locations with role assignments and permissions.

6 Conclusion

In this paper, we have proposed a location-based access control model that is based on RBAC. This model will be useful for pervasive computing applications in which the role of the user as well as his location will be used to determine if the user has access to some resource. We have shown how the different components of the core RBAC model are related with location, what new operations are needed and how existing operations must be changed to perform location-based access. We have formalized our model using the Z specification language. In future, we would like to extend this model by taking into account the effects of the constraints imposed by role hierarchy, static separation of duty, and dynamic separation of duty. We also plan to propose mechanisms by which such a model can be implemented. Typically location of a user or an object changes with time. Our future plans include proposing a model that takes into account these kinds of temporal constraints.

References

1. Elisa Bertino, Barbara Catania, Maria Luisa Damiani, and Paolo Perlasca. GEO-RBAC: A Spatially Aware RBAC. In *Proceedings of the ACM Symposium on Access Control Models and Technologies*, Stockholm, Sweden, June 2005.
2. Michael J. Covington, Prahlad Fogla, Zhiyuan Zhan, and Mustaque Ahamad. A Context-Aware Security Architecture for Emerging Applications. In *Proceedings of the Annual Computer Security Applications Conference* , pages 249–260, Las Vegas, NV, USA, December 2002.
3. Michael J. Covington, Wende Long, Srividhya Srinivasan, Anind Dey, Mustaque Ahamad, and Gregory Abowd. Securing Context-Aware Applications Using Environment Roles. In *Proceedings of the 6th ACM Symposium on Access Control Models and Technologies*, pages 10–20, Chantilly, VA, USA, May 2001.
4. Dorothy E. Denning and Peter F. MacDoran. Location-Based Authentication:Grounding Cyberspace for Better Security. In *Computer Fraud and Security,Elsevier Science Ltd*, February 1996.
5. Urs Hengartner and Peter Steenkiste. Implementing Access Control to People Location Information. In *Proceeding of the SACMAT'04 Yorktown Heights,California,USA*, June 2004.
6. Ulf Leonhardt and Jeff Magee. Security Consideration for a Distributed Location Service. *Imperial College of Science, Technology and Medicine, London, UK*, 1997.
7. B. Potter, J. Sinclair, and D. Till. *An Introduction to Formal Specification and Z.* Prentice-Hall, New York, NY, 1991.
8. Indrakshi Ray and Lijun Yu. Short Paper: Towards a Location-Aware Role-Based Access Control Model. In *Proceedings of the IEEE Conference on Security and Privacy for Emerging Areas in Communications Network*, Athens, Greece, September 2005.
9. Geetanjali Sampemane, Prasad Naldurg, and Roy H. Campbell. Access Control for Active Spaces. In *Proceedings of the Annual Computer Security Applications Conference* , pages 343–352, Las Vegas, NV, USA, December 2002.