A Spatio-Temporal Access Control Model Supporting Delegation for Pervasive Computing Applications*

Indrakshi Ray and Manachai Toahchoodee

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

Abstract. The traditional access control models, such as Role-Based Access Control (RBAC) and Bell-LaPadula (BLP), are not suitable for pervasive computing applications which typically lack well-defined security perimeters and where all the entities and interactions are not known in advance. We propose an access control model that handles such dynamic applications and uses environmental contexts to determine whether a user can get access to some resource. Our model is based on RBAC because it simplifies role management and is the de facto access control model for commercial organizations. However, unlike RBAC, it uses information from the environmental contexts to determine access decisions. The model also supports delegation which is important for dynamic applications where a user is unavailable and permissions may have to be transferred temporarily to another user/role in order to complete a specific task. This model can be used for any application where spatial and temporal information of a user and an object must be taken into account before granting access or temporarily transferring access to another user.

1 Introduction

With the increase in the growth of wireless networks and sensor and mobile devices, we are moving towards an era of pervasive computing. The growth of this technology will spawn applications such as, the Aware Home [6] and CMU's Aura [7], that will make life easier for people. However, before such applications can be widely deployed, it is important to ensure that no authorized users can access the resources of the application and cause security and privacy breaches. Traditional access control models, such as, Bell-LaPadula (BLP) and Role-Based Access Control (RBAC), do not work well for pervasive computing applications because they do not have well-defined security perimeters and all the users and resources are not known in advance. Moreover, they do not take into account environmental factors, such as, location and time, while making access decisions. Consequently, new access control models are needed for pervasive computing applications.

In pervasive computing applications, the access decisions cannot be based solely on the attributes of users and resources. For instance, we may want access to a computer be

* This work was supported in part by AFOSR under contract number FA9550-07-1-0042.

enabled when a user enters a room and it to be disabled when he leaves the room. Such types of access control can only be provided if we take environmental contexts, such as, time and location, into account before making access decisions. Thus, the access control model for pervasive computing applications must allow for the specification and checking of environmental conditions.

Pervasive computing applications are dynamic in nature and the set of users and resources are not known in advance. It is possible that a user/role for doing a specific task is temporarily unavailable and another user/role must be granted access during this time to complete it. This necessitates that the model be able to support delegation. Moreover, different types of delegation needs to be supported because of the unpredictability of the application.

Researchers have proposed various access control models for pervasive computing applications. Several works exist that focus on how RBAC can be extended to make it context aware [6, 5, 15]. Other extensions to RBAC include the Temporal Role-Based Access Control Model (TRBAC) [2] and the Generalized Temporal Role Based Access Control Model (GTRBAC) [9]. Researchers have also extended RBAC to incorporate spatial information [3, 14]. Incorporating both time and location in RBAC is also addressed by other researchers [1, 4, 12, 16]. Location-based access control has been addressed in other works not pertaining to RBAC [7, 8, 10, 13, 11, 17]. However, none of these works focus on delegation which is a necessity in pervasive computing applications.

In this paper, we propose a formal access control model for pervasive computing applications. The model extends the one proposed in our earlier work [12]. Since RBAC is policy-neutral, simplifies access management, and widely used by commercial applications, we base our work on it. We show how RBAC can be extended to incorporate environmental contexts, such as time and location. We illustrate how each component of RBAC is related with time and location and show how it is affected by them. We also show how spatio-temporal information is used for making access decisions. We also describe the different types of delegation that are supported by our model. Some of these are constrained by temporal and spatial conditions. The correct behavior of this model is formulated in terms of constraints that must be satisfied by any application using this model.

2 Our Model

Representing Location

There are two types of locations: *physical* and *logical*. All users and objects are associated with locations that correspond to the physical world. These are referred to as the physical locations. A *physical location* $PLoc_i$ 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. Physical locations are grouped into symbolic representations that will be used by applications. We refer to these symbolic representations as logical locations. Examples of logical locations are Fort Collins, Colorado etc. A *logical location* is an abstract notion for one or more physical locations. We assume the existence of two translation functions, m and m', that convert from logical locations to physical locations and vice-versa.

Although different kinds of relationships may exist between a pair of locations, here we focus only on *containment* relation. A physical location $ploc_j$ is said to be *contained* in another physical location $ploc_k$, denoted as, $ploc_j \subseteq ploc_k$, if the following condition holds: $\forall p_i \in ploc_j, p_i \in ploc_k$. The location $ploc_j$ is called the contained location and $ploc_k$ is referred to as the containing or the enclosing location. A logical location $lloc_m$ is contained in $lloc_n$, denoted as, $lloc_m \subseteq lloc_n$, if and only if the physical location corresponding to $lloc_m$ is contained within that of $lloc_m$, that is $m'(lloc_m) \subseteq m'(lloc_n)$. We assume the existence of a logical location called *universe* that contains all other locations. In the rest of the paper, we do not discuss physical locations any more. The locations referred to are logical locations.

Representing Time

A *time instant* is one discrete point on the time line. The exact granularity of a time instant will be application dependent. For instance, in some application a time instant may be measured at the nanosecond level and in another one it may be specified at the millisecond level. A *time interval* is a set of time instances. Example of an interval is 9:00 a.m. to 3:00 p.m. on 25th December. Example of another interval is 9:00 a.m. to 6:00 p.m. on Mondays to Fridays in the month of March. We use the notation $t_i \in d$ to mean that t_i is a time instance in the time interval d. A special case of relation between two time intervals that we use is referred to as *containment*. A time interval tv_i is contained in another interval tv_j, denoted as $tv_i \subseteq tv_j$, if the set of time instances in tv_i is a subset of those in tv_j. We introduce a special time interval, which we refer to as *always*, that includes all other time intervals.

Relationship of Core-RBAC entities with Location and Time

We discuss how the different entities of core RBAC, namely, *Users*, *Roles*, *Sessions*, *Permissions*, *Objects* and *Operations*, are associated with location and time.

Users

We assume that each valid user carries a locating device which is able to track his location. The location of a user changes with time. The relation $UserLocation(u,t)$ gives the location of the user at any given time instant t. Since a user can be associated with only one location at any given point of time, we have the following constraint:
$UserLocation(u,t) = l_i \wedge UserLocation(u,t) = l_j \Leftrightarrow (l_i \subseteq l_j) \vee (l_j \subseteq l_i)$
We define a similar function $UserLocations(u,d)$ that gives the location of the user during the time interval d. Note that, a single location can be associated with multiple users at any given point of time.

Objects

Objects can be physical or logical. Example of a physical object is a computer. Files are examples of logical objects. Physical objects have devices that transmit their location information with the timestamp. Logical objects are stored in physical objects. The location and timestamp of a logical object corresponds to the location and time of the physical object containing the logical object. We assume that each object is associated with one location at any given instant of time. Each location can be associated with many objects. The function *ObjLocation(o,t)* takes as input an object o and a time instance t and returns the location associated with the object at time t. Similarly, the function *ObjLocations(o,d)* takes as input an object o and time interval d and returns the location associated with the object.

Roles

We have three types of relations with roles. These are user-role assignment, user-role activation, and permission-role assignment. We begin by focusing on user-role assignment. Often times, the assignment of user to roles is location and time dependent. For instance, a person can be assigned the role of U.S. citizen only in certain designated locations and at certain times only. To get the role of conference attendee, a person must register at the conference location during specific time intervals. Thus, for a user to be assigned a role, he must be in designated locations during specific time intervals. In our model, a user must satisfy spatial and temporal constraints before roles can be assigned. We capture this with the concept of *role allocation*. A role is said to be *allocated* when it satisfies the temporal and spatial constraints needed for role assignment. A role can be assigned once it has been allocated. $RoleAllocLoc(r)$ gives the set of locations where the role can be allocated. $RoleAllocDur(r)$ gives the time interval where the role can be allocated. Some role s can be allocated anywhere, in such cases $RoleAllocLoc(s) = universe$. Similarly, if role p can be assigned at any time, we specify $RoleAllocDur(p) = always$.

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 when the show is on. The role of conference attendee can be activated only if the user is in the conference site while the conference is in session. In short, the user must satisfy temporal and location constraints before a role can be activated. We borrow the concept of *role-enabling* [2, 9] to describe this. A role is said to be *enabled* if it satisfies the temporal and location constraints needed to activate it. A role can be activated only if it has been enabled. $RoleEnableLoc(r)$ gives the location where role r can be activated and $RoleEnableDur(r)$ gives the time interval when the role can be activated.

The predicate $UserRoleAssign(u,r,d,l)$ states that the user u is assigned to role r during the time interval d and location l. For this predicate to hold, the location of the user when the role was assigned must be in one of the locations where the role allocation can take place. Moreover, the time of role assignment must be in the interval when role allocation can take place.

$UserRoleAssign(u,r,d,l) \Rightarrow$

$$(UserLocation(u,d) = l) \wedge (l \subseteq RoleAllocLoc(r)) \wedge (d \subseteq RoleAllocDur(r))$$

The predicate $UserRoleActivate(u,r,d,l)$ is true if the user u activated role r for the interval d at location l. This predicate implies that the location of the user during the role activation must be a subset of the allowable locations for the activated role and all times instances when the role remains activated must belong to the duration when the role can be activated and the role can be activated only if it is assigned.

$UserRoleActivate(u,r,d,l) \Rightarrow$

$$(l \subseteq RoleEnableLoc(r)) \wedge (d \subseteq RoleEnableDur(r)) \wedge UserRoleAssign(u,r,d,l)$$

The additional constraints imposed upon the model necessitates changing the preconditions of the functions *AssignRole* and *ActivateRole*. The permission role assignment is discussed later.

Sessions

In mobile computing or pervasive computing environments, we have different types of sessions that can be initiated by the user. Some of these sessions can be location-

dependent, others not. Thus, sessions are classified into different types. Each instance of a session is associated with some type of a session. The type of session instance s is given by the function $Type(s)$. The type of the session determines the allowable location. The allowable location for a session type st is given by the function $SessionLoc(st)$. When a user u wants to create a session si, the location of the user for the entire duration of the session must be contained within the location associated with the session. The predicate $SessionUser(u,s,d)$ indicates that a user u has initiated a session s for duration d.

$SessionUser(u,s,d) \Rightarrow (UserLocation(u,d) \subseteq SessionLoc(Type(s)))$

Since sessions are associated with locations, not all roles can be activated within some session. The predicate $SessionRole(u,r,s,d,l)$ states that user u initiates a session s and activates a role for duration d and at location l.

$SessionRole(u,r,s,d,l) \Rightarrow UserRoleActivate(u,r,d,l) \wedge l \subseteq SessionLoc(Type(s)))$

Permissions

Our model also allows us to model real-world requirements where access decision is contingent upon the time and location associated with the user and the object. For example, a teller may access the bank confidential file if and only if he is in the bank and the file location is the bank secure room and the access is granted only during the working hours. Our model should be capable of expressing such requirements.

Permissions are associated with roles, objects, and operations. We associate three additional entities with permission to deal with spatial and temporal constraints: user location, object location, and time. We define three functions to retrieve the values of these entities. $PermRoleLoc(p,r)$ specifies the allowable locations that a user playing the role r must be in for him to get permission p. $PermObjLoc(p,o)$ specifies the allowable locations that the object o must be in so that the user has permission to operate on the object o. $PermDur(p)$ specifies the allowable time when the permission can be invoked.

We define another predicate which we term *PermRoleAcquire(p,r,d,l)*. This predicate is true if role r has permission p for duration d at location l. Note that, for this predicate to be true, the time interval d must be contained in the duration where the permission can be invoked and the role can be enabled. Similarly, the location l must be contained in the places where the permission can be invoked and role can be enabled.

$PermRoleAcquire(p,r,d,l) \Rightarrow$

$(l \subseteq (PermRoleLoc(p,r) \cap RoleEnableLoc(r))) \wedge (d \subseteq (PermDur(p) \cap RoleEnableDur(p)))$

The predicate $PermUserAcquire(u,o,p,d,l)$ means that user u can acquire the permission p on object o for duration d at location l. This is possible only when the permission p is assigned some role r which can be activated during d and at location l, the user location and object location match those specified in the permission, the duration d matches that specified in the permission.

$PermRoleAcquire(p,r,d,l) \wedge UserRoleActivate(u,r,d,l)$

$\wedge (ObjectLocation(o,d) \subseteq PermObjectLoc(p,o)) \Rightarrow PermUserAcquire(u,o,p,d,l)$

For lack of space, we do not discuss the impact of time and location on role-hierarchy or separation of duty, but refer the interested reader to one of our earlier paper [12].

Impact of Time and Location on Delegation

Many situations require the temporary transfer of access rights to accomplish a given task. For example, in a pervasive computing application, a doctor may give certain privilege to a trained nurse, when he is taking a short break. In such situations, the doctor can give a subset of his permissions to the nurse for a given period of time. There are a number of different types of delegation. The entity that transfers his privileges temporarily to another entity is often referred to as the delegator. The entity who receives the privilege is known as the delegatee. The delegator (delegatee) can be either an user or a role. Thus, we may have four types of delegations: *user to user* (U2U), *user to role* (U2R), *role to role* (R2R), and *role to user* (R2U). System administrators are responsible for overseeing delegation when the delegator is a role. Individual users administer delegation when the delegator is an user. When a user is the delegator, he can delegate a subset of permissions that he possesses by virtue of being assigned to different roles. When a role is the delegator, he can delegate either a set of permissions or he can delegate the entire role. We can therefore classify delegation on the basis of role delegation or permission delegation. We identify the following types of delegation.

[U2U Unrestricted Permission Delegation] In this type of delegation, the delegatee can invoke the delegator's permissions at any time and at any place where the delegator could invoke those permissions. The illness of the company president caused him to delegate his email reading privilege to his secretary.

Let $DelegateU2U\_P_u(u, v, Perm)$ be the predicate that allows user u to delegate the permissions in the set $Perm$ to user v without any temporal or spatial constraints. This will allow v to invoke the permissions at any time or at any place.

$\forall p \in Perm, DelegateU2U\_P_u(u, v, Perm) \wedge PermUserAcquire(u, o, p, d, l) \Rightarrow$
$PermUserAcquire(v, o, p, d, l)$

[U2U Time Restricted Permission Delegation] Here the delegator places time restrictions on when the delegatee can invoke the permissions. However, no special restrictions are placed with respect to location – the delegatee can invoke the permission at any place that the delegator could do so. The professor can delegate his permission to proctor an exam to the teaching assistant while he is on travel.

Let $DelegateU2U\_P_t(u, v, Perm, d')$ be the predicate that allows user u to delegate the permissions in the set $Perm$ to user v for the duration d'.

$\forall p \in Perm, DelegateU2U\_P_t(u, v, Perm, d') \wedge PermUserAcquire(u, o, p, d, l) \wedge (d' \subseteq d)$
$\Rightarrow PermUserAcquire(v, o, p, d', l)$

[U2U Location Restricted Permission Delegation] A delegator can place spatial restrictions on when the delegatee can invoke the permissions. However, the only temporal restriction is that the delegatee can invoke the permissions during the period when the original permission is valid. The teaching assistant can delegate the permission regarding lab supervision to the lab operator only in the Computer Lab.

Let $DelegateU2U\_P_l(u, v, Perm, l')$ be the predicate that allows user u to delegate the permissions in the set $Perm$ to user v in the location l'.

$\forall p \in Perm, DelegateU2U\_P_l(u, v, Perm, l') \wedge PermUserAcquire(u, o, p, d, l) \wedge (l' \subseteq l)$
$\Rightarrow PermUserAcquire(v, o, p, d, l')$

[U2U Time Location Restricted Permission Delegation] In this case, the delegator imposes a limit on the time and the location where the delegatee can invoke the permis-

sion. A nurse can delegate his permission to oversee a patient while he is resting in his room to a relative.

Let $DelegateU2U\_P_{tl}(u, v, Perm, d', l')$ be the predicate that allows user u to delegate the permissions in the set $Perm$ to user v in the location l' for the duration d'.

$\forall p \in Perm, DelegateU2U\_P_{tl}(u, v, Perm, t', l') \wedge PermUserAcquire(u, o, p, d, l)$
$\wedge (d' \subseteq d) \wedge (l' \subseteq l) \Rightarrow PermUserAcquire(v, o, p, d', l')$

[U2U Unrestricted Role Delegation] Here the delegator delegates a role to the delegatee. The delegatee can activate the roles at any time and place where the delegator can activate those roles. A manager before relocating can delegate his roles to his successor in order to train him.

Let $DelegateU2U\_R_u(u, v, r)$ be the predicate that allows user u to delegate his role r to user v.

$DelegateU2U\_R_u(u, v, r) \wedge UserRoleActivate(u, r, d, l) \Rightarrow UserRoleActivate(v, r, d, l)$

[U2U Time Restricted Role Delegation] In this case, the delegator delegates a role to the delegatee but the role can be activated only for a more limited duration than the original role. A user can delegate his role as a teacher to a responsible student while he is in a conference.

Let $DelegateU2U\_R_t(u, v, r, d')$ be the predicate that allows user u to delegate his role r to user v for the duration d'.

$DelegateU2U\_R_t(u, v, r, d') \wedge UserRoleActivate(u, r, d, l) \wedge$
$(d' \subseteq RoleEnableDur(r)) \wedge (d' \subseteq d) \Rightarrow UserRoleActivate(v, r, d', l)$

[U2U Location Restricted Role Delegation]: In this case, the delegator delegates a role to the delegatee but the role can be activated in more limited locations than the original role. A student can delegate his lab supervision role to another student in a designated portion of the lab only.

Let $DelegateU2U\_R_l(u, v, r, l')$ be the predicate that allows user u to delegate his role r to user v in the location l'.

$Delegate\_R_l(u, v, r, l') \wedge UserRoleActivate(u, r, d, l) \wedge$
$(l' \subseteq RoleEnableLoc(r)) \wedge (l' \subseteq l) \Rightarrow UserRoleActivate(v, r, d, l')$

[U2U Time Location Restricted Role Delegation] The delegator delegates the role, but the delegatee can activate the role for a limited duration in limited places. A student can delegate his lab supervision role to another student only in the lab when he leaves the lab for emergency reasons.

Let $DelegateU2U\_R_{tl}(u, v, r, d', l')$ be the predicate that allows user u to delegate his role r to user v in location l' and time d'.

$DelegateU2U\_R_{tl}(u, v, r, d', l') \wedge UserRoleActivate(u, r, d, l) \wedge (l' \subseteq RoleEnableLoc(r)) \wedge$
$(d' \subseteq RoleEnableDur(r)) \wedge (d' \subseteq d) \wedge (l' \subseteq l) \Rightarrow UserRoleActivate(v, r, d', l')$

[R2R Unrestricted Permission Delegation] Here, all users assigned to the delegatee role can invoke the delegator role's permissions at any time and at any place where the user of this delegator role could invoke those permissions. The Smart Home owner role may delegate the permission to check the status of security sensors of the home to the police officer role, so all police officers can detect the intruder at any time at any place.

Let $DelegateR2R\_P_u(r_1, r_2, Perm)$ be the predicate that allows role r_1 to delegate the permissions in the set $Perm$ to role r_2 without any temporal or spatial constraints. This will allow users in the role r_2 to invoke the permissions at any time or at any place.

$\forall p \in Perm, DelegateR2R\_P_u(r_1, r_2, Perm) \wedge PermRoleAcquire(p, r_1, d, l) \wedge$
$(d \subseteq RoleEnableDur(r_2)) \wedge (l \subseteq RoleEnableLoc(r_2))$
$\Rightarrow PermRoleAcquire(p, r_2, d, l)$

[R2R Time Restricted Permission Delegation] The delegator role can place temporal restrictions on when the users of the delegatee role can invoke the permissions. No special restrictions are placed with respect to location i.e. the delegatee role's users can invoke the permissions at any place that the delegator role's users could do so. CS599 teacher role can grant the permission to access course materials to CS599 student role for the specific semester.

Let $DelegateR2R\_P_t(r_1, r_2, Perm, d')$ be the predicate that allows role r_1 to delegate the permissions in the set $Perm$ to role r_2 for the duration d'.

$\forall p \in Perm, DelegateR2R\_P_t(r_1, r_2, Perm, d') \wedge (d' \subseteq d)\ PermRoleAcquire(p, r_1, d, l) \wedge$
$(l' \subseteq l) \wedge (d' \subseteq RoleEnableDur(r_2)) \wedge (l \subseteq RoleEnableLoc(r_2))$
$\Rightarrow PermRoleAcquire(p, r_2, d', l)$

[R2R Location Restricted Permission Delegation] Here, the delegator role places spatial constraints on where the users of the delegatee role can invoke the permissions. No special temporal constraints are placed, that is, the delegatee role's users can invoke the permissions at any time that the delegator role's users could do so. The librarian role may grant the permission to checkout the book to the student role only at the self-checkout station.

Let $DelegateR2R\_P_l(r_1, r_2, Perm, l')$ be the predicate that allows role r_1 to delegate the permissions in the set $Perm$ to role r_2 in the location l'.

$\forall p \in Perm, DelegateR2R\_P_l(r_1, r_2, Perm, l') \wedge PermRoleAcquire(p, r_1, d, l) \wedge$
$(d \subseteq RoleEnableDur(r_2)) \wedge (l' \subseteq RoleEnableLoc(r_2)) \wedge (l' \subseteq l)$
$\Rightarrow PermRoleAcquire(p, r_2, d, l')$

[R2R Time Location Restricted Permission Delegation] Here the delegator role imposes a limit on the time and the location where the delegatee role's users could invoke the permissions. The daytime doctor role may delegate the permission to get his location information to the nurse role only when he is in the hospital during the daytime.

Let $DelegateR2R\_P_{tl}(r_1, r_2, Perm, d', l')$ be the predicate that allows role r_1 to delegate the permissions in the set $Perm$ to role r_2 in the location l' for the duration d'.

$\forall p \in Perm, DelegateR2R\_P_{tl}(r_1, r_2, Perm, d', l') \wedge PermRoleAcquire(p, r_1, d, l) \wedge$
$(d' \subseteq RoleEnableDur(r_2)) \wedge (l' \subseteq RoleEnableLoc(r_2)) \wedge (d' \subseteq d) \wedge (l' \subseteq l)$
$\Rightarrow PermRoleAcquire(p, r_2, d', l')$

[R2R Unrestricted Role Delegation] Here all users assigned to the delegatee role can activate the delegator role at any time and at any place where the user of this delegator role could activate the role. In the case of company reorganization, the manager role can be delegated to the manager successor role in order to train him.

Let $DelegateR2R\_R_u(r_1, r_2)$ be the predicate that allows role r_1 to be delegated to role r_2.

$DelegateR2R\_R_u(r_1, r_2) \wedge UserRoleActivate(u, r_2, d, l) \wedge (d \subseteq RoleEnableDur(r_1)) \wedge$
$(l \subseteq RoleEnableLoc(r_1)) \Rightarrow UserRoleActivate(u, r_1, d, l)$

[R2R Time Restricted Role Delegation] Here, the delegator places temporal constraints on when the users of the delegatee role can activate the delegator role. No special spatial constraints are placed i.e. the delegatee role's users can activate the del-

egator role at any place that the delegator role's users could do so. The permanent staff role can be delegated to the contract staff role during the contract period.

Let $DelegateR2R\_R_t(r_1, r_2, d')$ be the predicate that allows role r_1 to be delegated to role r_2 for the duration d'.

$$DelegateR2R\_R_t(r_1, r_2, d') \wedge UserRoleActivate(u, r_2, d', l) \wedge (d \subseteq RoleEnableDur(r_1)) \wedge (l \subseteq RoleEnableLoc(r_1)) \wedge (d' \subseteq d) \Rightarrow UserRoleActivate(u, r_1, d', l)$$

[R2R Location Restricted Role Delegation] The delegator role can place spatial restrictions on where the users of the delegatee role can activate the delegator role. No special restrictions are placed with respect to time i.e. the delegatee role's users can activate the delegator role at any time that the delegator role's users could do so. The researcher role can be delegated to the lab assistant role at the specific area of the lab.

Let $DelegateR2R\_R_l(r_1, r_2, l')$ be the predicate that allows role r_1 to be delegated to role r_2 in the location l'.

$$DelegateR2R\_R_l(r_1, r_2, l') \wedge UserRoleActivate(u, r_2, d, l') \wedge (d \subseteq RoleEnableDur(r_1)) \wedge (l \subseteq RoleEnableLoc(r_1)) \wedge (l' \subseteq l) \Rightarrow UserRoleActivate(u, r_1, d, l')$$

[R2R Time Location Restricted Role Delegation] In this case, the delegator role imposes a limit on the time and the location where the delegatee role's users could activate the role. The full-time researcher role can be delegated to the part-time researcher role only during the hiring period in the specific lab.

Let $DelegateR2R\_R_{tl}(r_1, r_2, d', l')$ be the predicate that allows role r_1 to be delegated to role r_2 in the location l' for the duration d'.

$$DelegateR2R\_R_{tl}(r_1, r_2, d', l') \wedge UserRoleActivate(u, r_2, d', l') \wedge (d' \subseteq d) \wedge (l' \subseteq l) \wedge (d \subseteq RoleEnableDur(r_1)) \wedge (l \subseteq RoleEnableLoc(r_1)) \wedge (d' \subseteq d) \wedge (l' \subseteq l) \Rightarrow UserRoleActivate(u, r_1, d', l')$$

3 Conclusion and Future Work

Traditional access control models which do not take into account environmental factors before making access decisions may not be suitable for pervasive computing applications. Towards this end, we proposed a spatio-temporal role based access control model that supports delegation. The behavior of the model is formalized using constraints.

An important work that we plan to do is the analysis of the model. We have proposed many different constraints. We are interested in finding conflicts and redundancies among the constraint specification. Such analysis is needed before our model can be used for real world applications. We plan to investigate how to automate this analysis. We also plan to implement our model. We need to investigate how to store location and temporal information and how to automatically detect role allocation and enabling using triggers.

References

1. Vijayalakshmi Atluri and Soon Ae Chun. A geotemporal role-based authorisation system. *International Journal of Information and Computer Security*, 1(1/2):143–168, January 2007.
2. Elisa Bertino, Piero Andrea Bonatti, and Elena Ferrari. TRBAC: A Temporal Role-Based Access Control Model. In *Proceedings of the 5th ACM workshop on Role-Based Access Control*, pages 21–30, Berlin, Germany, July 2000. ACM Press.

3. Elisa Bertino, Barbara Catania, Maria Luisa Damiani, and Paolo Perlasca. GEO-RBAC: a spatially aware RBAC. In *Proceedings of the 10th ACM Symposium on Access Control Models and Technologies*, pages 29–37, Stockholm, Sweden, June 2005. ACM Press.
4. Suroop Mohan Chandran and J. B. D. Joshi. *LoT-RBAC*: A Location and Time-Based RBAC Model. In *WISE*, pages 361–375, 2005.
5. 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.
6. 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.
7. Urs Hengartner and Peter Steenkiste. Implementing Access Control to People Location Information. In *Proceeding of the 9th Symposium on Access Control Models and Technologies*, Yorktown Heights, New York, June 2004.
8. R. J. Hulsebosch, A. H. Salden, M. S. Bargh, P. W. G. Ebben, and J. Reitsma. Context sensitive access control. In *Proceedings of the 10th ACM Symposium on Access Control Models and Technologies*, pages 111–119, Stockholm, Sweden, 2005. ACM Press.
9. James B.D. Joshi, Elisa Bertino, Usman Latif, and Arif Ghafoor. A Generalized Temporal Role-Based Access Control Model. *IEEE Transactions on Knowledge and Data Engineering*, 17(1):4–23, 2005.
10. Ulf Leonhardt and Jeff Magee. Security Consideration for a Distributed Location Service. *Imperial College of Science, Technology and Medicine, London, UK*, 1997.
11. Fang Pu, Daoqin Sun, Qiying Cao, Haibin Cai, and Fan Yang. Pervasive Computing Context Access Control Based on $UCON_{ABC}$ Model. In *Intelligent Information Hiding and Multimedia Signal Processing, 2006. IIH-MSP '06. International Conference on*, pages 689–692, December 2006.
12. I. Ray and M. Toahchoodee. A Spatio-Temporal Role-Based Access Control Model. In *Proceedings of the 21st Annual IFIP WG 11.3 Working Conference on Data and Applications Security*, pages 211–226, Redondo Beach, CA, July 2007.
13. Indrakshi Ray and Mahendra Kumar. Towards a Location-Based Mandatory Access Control Model. *Computers & Security*, 25(1), February 2006.
14. Indrakshi Ray, Mahendra Kumar, and Lijun Yu. LRBAC: A Location-Aware Role-Based Access Control Model. In *Proceedings of the 2nd International Conference on Information Systems Security*, pages 147–161, Kolkata, India, December 2006.
15. 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.
16. Arjmand Samuel, Arif Ghafoor, and Elisa Bertino. A Framework for Specification and Verification of Generalized Spatio-Temporal Role Based Access Control Model. Technical report, Purdue University, February 2007. CERIAS TR 2007-08.
17. Hai Yu and Ee-Peng Lim. LTAM: A Location-Temporal Authorization Model. In *Secure Data Management*, pages 172–186, 2004.