Using Graph Theory to Represent a Spatio-Temporal Role-Based Access Control Model

MANACHAI TOAHCHOODEE
Colorado State University
INDRAKSHI RAY
Colorado State University
and
ROSS M. MCCONNELL
Colorado State University

Pervasive computing applications have unique characteristics that preclude the use of traditional access control models, such as Role-Based Access Control (RBAC), for their protection. Such models do not take into account contextual information before making access decisions and cannot handle the dynamism inherent in pervasive computing applications. Towards this end, we propose new spatio-temporal access control models for use in such applications and formally define their semantics using graph-theoretic notation. The dynamism inherent in pervasive computing applications may cause the access control configuration to change while the policies are deployed. Towards this end, we show how to perform incremental analysis to give assurance that security breaches do not occur as a result of changing the access control configuration. Our analysis makes clever use of data structures and achieves good time complexity results.

Categories and Subject Descriptors: D.4.6 [**Security and Protection**]: Access controls; G.2.2 [**Graph Theory**]: Graph algorithms

General Terms: Access Control Models, Pervasive Computing, Algorithms

Additional Key Words and Phrases: Access Control Models, Pervasive Computing, Access Control Models Analysis, Graph Algorithms

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 [Covington et al. 2001] and CMU's Aura [Hengartner and Steenkiste 2004], that will make life easier for people. However, before such applications can be widely deployed, it is important to ensure that no authorized user 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), works well for applications having a well-defined security perimeter, which is often not the case for pervasive computing applications. 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 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 it is possible that a user/role that typically does 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 need to be supported because of the unpredictability of the application.

Researchers have proposed spatio-temporal access control models that can be used in pervasive computing applications [Chandran and Joshi 2005; Ray and Toahchoodee 2007; Toahchoodee and Ray 2008; Samuel et al. 2007]. Since RBAC is policy-neutral, simplifies access management, and widely used by commercial applications, most of the works are based on it. The models are specified in terms of constraints that support the various features of the model whose analysis is a non-trivial problem. We propose a spatio-temporal RBAC model that supports delegation. Our model has a well-defined semantics which we express using graph-theoretic notations. The use of graph-theory also allows one to visualize the relationships and interactions among the different components in the model. Using the directed graph representation, the interaction and relationship between components in the model becomes clearer and more expressive. It also allows one to detect the presence of inconsistencies. The result is a simple spatio-temporal access control model that can be used for real-world applications.

Pervasive computing applications are dynamic in nature. While the application is executing, the entities requiring access or the resources needing protection may change. In the face of such dynamism, it is essential to ensure that access control breaches do not occur. Since the analysis must be done in real-time, it is important to minimize the verification time. Towards this end, we provide techniques for incremental analysis with good time complexity results. For example, to detect separation of duty (SoD) violations in a dynamic graph, we need to find whether the nodes connected by SoD constraints have a common predecessor. Applying a naive algorithm based on Depth First Search, requires $O(kE)$ time for each change applied to the graph, where k is the number of SoD constraints and E is the number of edges. We improve upon this result significantly by proposing a new common predecessor detection algorithm in a dynamic graph.

The rest of the paper is organized as follows. Section 2 describes the related work. Section 3 specifies our model using a graphical representation. Sections 4 and 5 show how we represent the different types of separation of duty constraints and delegation operations that we can have in our model. Section 6 formalizes important properties of the graph and discusses how they can be verified in the face of access control configuration changes. Section 7 illustrates the features of our model using a simple example. Section 8 concludes the paper with some pointers to future directions. Appendix presents some of our results in graph theory which is needed for this work.

2. RELATED WORK

Location-based access control has been addressed in works not pertaining to RBAC [Ardagna et al. 2006; Atluri and Chun 2004; 2007; Hengartner and Steenkiste 2004; Hulsebosch et al. 2005; Leonhardt and Magee 1997; Pu et al. 2006; Ray and Kumar 2006; Yu and Lim 2004]. Atluri and Chun proposed the Geospatial Data Authorization Model (GSAM) which is an authorization model for managing geospatial information [Atluri and Chun 2004; 2007].

Our work is based on RBAC [Ferraiolo et al. 2001] which is often used for addressing the access control needs of commercial organizations. Researchers have also extended RBAC to support delegation which causes temporary transfer or grant of access privileges from the delegator to delegatee [Barka and Sandhu 2000a; 2000b; 2004; Crampton and Khambhammettu 2008; Zhang et al. 2003]. Researchers have also worked on extending RBAC to support pervasive computing applications [Covington et al. 2002; Covington et al. 2001; Sampemane et al. 2002]. Sampemane et al. present a new access control model for active spaces which denote the computing environment integrating physical spaces and embedded computing software and hardware entities [Sampemane et al. 2002]. Covington et al. introduce environment roles in a generalized RBAC model (GRBAC) to help control access to private information and resources in ubiquitous computing applications [Covington et al. 2001]. In a subsequent work [Covington et al. 2002], Covington et al. describe the Context-Aware Security Architecture (CASA) which is an implementation of the GRBAC model.

Other extensions to RBAC include the Temporal Role-Based Access Control Model (TRBAC) proposed by Bertino et al. [Bertino et al. 2000] that adds the time dimension to the RBAC model. Joshi et al. extend this work by proposing the Generalized Temporal Role Based Access Control Model (GTRBAC) that introduces the concept of time-based role hierarchy and time-based separation of duty [Joshi et al. 2005]. In another work, Joshi and Bertino extend the GTRBAC model to support fine-grained delegation [Joshi and Bertino 2006]. The formal analysis of the different types of time-based hybrid hierarchy is proposed by Joshi et al. in [Joshi et al. 2008]. Researchers have also extended RBAC to incorporate spatial information [Bertino et al. 2005; Ray et al. 2006]. Incorporating both time and location in RBAC has been addressed in several works [Chandran and Joshi 2005; Chen and Crampton 2008; Ray and Toahchoodee 2007; 2008; Samuel et al. 2007].

Nyanchama et al. [Nyanchama and Osborn 1999] present a graph-based reference model for role-based access control and demonstrate how hierarchies and separation of duties can be represented in this model. Chen and Crampton develop the graph based representation for the spatio-temporal RBAC in [Chen and Crampton 2008]. The RBAC entities are represented by vertices while their relationships are represented by the edges of a directed graph. The authors propose three types of models: standard, strong, and weak. We extend this model to support these necessary functionalities, such as, separation of duty constraints and delegation constraints and also illustrate how the model can be analyzed to detect the presence of inconsistencies. Many work appears that attempt to analyze RBAC specifications using existing formal languages and tools [Ray et al. 2006; Ray et al. 2004; Toahchoodee and Ray 2008; 2009; Yuan et al. 2006]. Some of these analysis are done manually

[Ray et al. 2006; Ray et al. 2004; Yuan et al. 2006], and in others the process of translation is non-trivial [Toahchoodee and Ray 2008; 2009].

Not much work appears in the context of verifying dynamic access control. We have shown that detecting SoD violation involves finding a common predecessor in a dynamic graph. Related work along these lines include the work by Franciosa et al. [Franciosa et al. 1997]. Franciosa et al [Franciosa et al. 1997] proposed an incremental algorithm to maintain a Depth-First-Search (DFS) forest in a directed acyclic graph (DAG) under a sequence of edge insertions. The algorithm can be performed in $O(VE)$, where V is the number of vertices and E is the total number of edges after the insertions. The work, however, does not provide the algorithm to find the common predecessor. Our work addresses this shortcoming.

3. STARBACD: OUR SPATIO-TEMPORAL MODEL

Our model extends the one proposed by Chen and Crampton [Chen and Crampton 2008] in the following ways. First, we believe that a spatio-temporal access control model must also support access control for moving objects, that is, objects whose physical location changes with time. Second, separation of duties must also be supported by access control models. Third, the model must also provide support for delegation which is an absolute necessity for access control in pervasive computing applications.

We first propose the model where access is dependent on the location of the user as well as that of the object. In other words, a user will be allowed to access an object only if the user and the object are in specific locations. In our work, the set of vertices $V = U \cup R \cup P \cup O$ correspond to the RBAC entities: Users (U), Roles (R), Permissions (P), and Objects (O). Our model assumes the existence of the following relationships of RBAC that constitute the set of edges $E = UA \cup PA \cup PO \cup RH_a \cup RH_u$ where

—User-Role Assignment $(UA) = U \times R$

—Permission-Role Assignment $(PA) = R \times P$

—Permission-Object Assignment $(PO) = P \times O$

—Role Hierarchy $(RH) = R \times R \times \{a, u\}$, which can be categorized to:
  —the activation hierarchy $(RH_a) = \{(r, r') : (r, r', a) \in RH\}$, and
  —the permission usage hierarchy $(RH_u) = \{(r, r') : (r, r', u) \in RH\}$

We define the notion of activation path, usage path and access path in a manner similar to that proposed by Chen and Crampton. An *activation path* (or *act-path*) between v_1 and v_n is defined to be a sequence of vertices $v_1, \ldots, v_n$ such that $(v_1, v_2) \in UA$ and $(v_{i-1}, v_i) \in RH_a$ for $i = 3, \ldots, n$. A *usage path* (or *u-path*) between v_1 and v_n is defined to be a sequence of vertices $v_1, \ldots, v_n$ such that $(v_i, v_{i+1}) \in RH_u$ for $i = 1, \ldots, n-2$, and $(v_{n-1}, v_n) \in PA$. An *access path* (or *acs-path*) between v_1 and v_n is defined to be a sequence of vertices $v_1, \ldots, v_n$, such that (v_1, v_i) is an act-path, (v_i, v_{n-1}) is an u-path, and $(v_{n-1}, v_n) \in PO$. Following Chen and Crampton's work [Chen and Crampton 2008], we assume the existence of a spatio-temporal domain $\mathcal{D}$. We also propose three models, namely, the standard model, the strong model, and the weak model. The models differ with respect to

the spatio-temporal constraints that must be satisfied by the entities for the authorization to be successful.

Authorization in the Standard Model STARBACD$^{=}$

In the standard model, the individual entities, namely, users, roles, permissions, and objects, are associated with set of points in the spatio-temporal domain. These points indicate when and where the individual entities can be activated. The spatio-temporal points associated with the user describe when and where the user can create a session, those associated with a role specify when and where the role can be activated, those associated with a permission state when and where a permission can be invoked, and those associated with an object state when and where the object can be accessed. The standard model requires that if a user u can access an object o at some spatio-temporal point d, then d is contained in the set of spatio-temporal points associated with all the nodes in the path connecting u to o. These ideas are formalized below.

The spatio-temporal constraints in the *standard STARBACD model* (or STARBACD$^{=}$) are denoted with a function $\lambda : V \rightarrow 2^{\mathcal{D}}$. For $v \in V$, $\lambda(v) \subseteq \mathcal{D}$ denotes the set of points in space-time at which v can be invoked.

—if $u \in U$, then $\lambda(u)$ denotes the set of points in space-time at which u may create a session;

—if $r \in R$, then $\lambda(r)$ denotes the set of points in space-time at which r may be activated in a session;

—if $p \in P$, then $\lambda(p)$ denotes the set of points in space-time at which p may be granted;

—if $o \in O$, then $\lambda(o)$ denotes the set of points in space-time at which o may be accessible.

Given a path $v_1, \ldots, v_n$ in the labeled graph $G = (V, E, \lambda)$, where $E = UA \cup PA \cup PO \cup RH_a \cup RH_u$, we write $\hat{\lambda}(v_1, \ldots, v_n) = \hat{\lambda}(v_1, v_n) \subseteq \mathcal{D}$ to denote $\bigcap_{i=1}^{n} \lambda(v_i)$. In other words, $\hat{\lambda}(v_1, v_n)$ is the set of points at which every vertex v_i is enabled. Note that semantics of λ and $\hat{\lambda}$ are consistent with those proposed by Chen and Crampton [Chen and Crampton 2008].

Authorization in STARBACD$^{=}$:.

. A user $v \in U$ may activate role $v' \in R$ at point $d \in \mathcal{D}$ if and only if there exists an act-path $v = v_1, v_2, \ldots, v_n = v'$ and $d \in \hat{\lambda}(v, v')$;

. A role $v \in R$ is authorized for permission $v' \in P$ at point $d \in \mathcal{D}$ if and only if there exists an u-path $v = v_1, v_2, \ldots, v_n = v'$ and $d \in \hat{\lambda}(v, v')$;

. A user $v \in U$ is authorized for permission $v' \in P$ with respect to object $v'' \in O$ at point $d \in \mathcal{D}$ if and only if there exists an acs-path $v = v_1, v_2, \ldots, v_i, \ldots, v_{n-1} = v', v_n = v''$ such that $v_i \in R$ for some i, $v_1, \ldots, v_i$ is an act-path, $v_i, \ldots, v_{n-1}$ is a u-path, $(v_{n-1}, n) \in PO$, and $d \in \hat{\lambda}(v, v'')$;

Authorization in the Strong Model STARBACD^{+}

The strong model is used when not only the individual entities (users, roles, permissions, objects) involved must satisfy the spatio-temporal constraints, but

the different relationships must also satisfy such constraints. For instance, consider the relation $(r, p) \in PA$. In this case, we not only have to take into account the spatio-temporal points at which the role r can be activated in a session and the points at which the permission p can be invoked, but we also must consider the spatio-temporal points when r can invoke p. This requires specifying another function in the strong model as described below.

The spatio-temporal constraints in the *strong STARBACD model* (or STARBACD$^+$) are denoted with a function $\mu : E \rightarrow 2^{\mathcal{D}}$. For $e = (v, v') \in E$, $\mu(v, v')$ denotes the set of points in space-time at which the association between v and v' is enabled.

—if $(u, r) \in UA$, then $\mu(u, r)$ denotes the set of points in space-time at which u is assigned to r;
—if $(r', r) \in RH_a$, then $\mu(r', r)$ denotes the set of points in space-time at which r' is senior to r in the activation hierarchy;
—if $(r', r) \in RH_u$, then $\mu(r', r)$ denotes the set of points in space-time at which r' is senior to r in the permission usage hierarchy;
—if $(r, p) \in PA$, then $\mu(r, p)$ denotes the set of points in space-time at which p is assigned to r.
—if $(p, o) \in PO$, then $\mu(p, o)$ denotes the set of points in space-time at which o is assigned to p.

Given a path $v_1, \ldots, v_n$ in the labeled graph $G = (V, E, \lambda, \mu)$, where $V = U \cup R \cup P \cup O$ and $E = UA \cup PA \cup PO \cup RH_a \cup RH_u$, we write $\hat{\mu}(v_1, \ldots, v_n) = \hat{\mu}(v_1, v_n) \subseteq \mathcal{D}$ to denote $\bigcap_{i=1}^{n-1} \mu(v_i, v_{i+1})$. The semantics imply that an edge can only be enabled if both endpoints are enabled. Hence, $\hat{\mu}(v_1, v_n)$ is the set of points at which every vertex and every edge in the path is enabled. Here again the semantics of μ and $\hat{\mu}$ are consistent with those proposed by Chen and Crampton [Chen and Crampton 2008].

Authorization in STARBACD$^+$:.

. a user $v \in U$ may activate role $v' \in R$ at point $d \in \mathcal{D}$ if and only if there exists an act-path $v = v_1, v_2, \ldots, v_n = v'$ and $d \in \hat{\mu}(v, v')$;

. a role $v \in R$ is authorized for permission $v' \in P$ at point $d \in \mathcal{D}$ if and only if there exists an u-path $v = v_1, v_2, \ldots, v_n = v'$ and $d \in \hat{\mu}(v, v')$;

. a user $v \in U$ is authorized for permission $v' \in P$ with respect to object $v'' \in O$ at point $d \in \mathcal{D}$ if and only if there exists an acs-path $v = v_1, v_2, \ldots, v_i, \ldots, v_{n-1} = v', v_n = v''$ such that $v_i \in R$ for some i, $v_1, \ldots, v_i$ is an act-path, $v_i, \ldots, v_{n-1}$ is an u-path, $(v_{n-1}, v_n) \in PO$ and $d \in \hat{\mu}(v, v'')$;

Authorization in the Weak Model STARBACD$^-$

The weak model is derived from the standard model. Recall that the standard model requires that each entity (users, roles, permissions, and objects) in the authorization path be associated with a set of spatio-temporal points and the intersections of all these sets be non-zero. In the weak model, the entity v is authorized for another entity v' provided there is overlap in their spatio-temporal points. There is no requirement that the intermediate nodes on the path satisfy the spatio-temporal constraints. Like STARBACD$^=$, the model is based on the labeled graph $G = (V, E, \lambda)$, where $V = U \cup R \cup P \cup O$ and $E = UA \cup PA \cup PO \cup RH_a \cup RH_u$.

Authorization in STARBACD$^-$:.

. A user $v \in U$ may activate role $v' \in R$ at point $d \in \mathcal{D}$ if and only if there exists an act-path $v = v_1, v_2, \ldots, v_n = v'$ and $d \in \lambda(v) \cap \lambda(v')$;

. A role $v \in R$ is authorized for permission $v' \in P$ at point $d \in \mathcal{D}$ if and only if there exists a u-path $v = v_1, v_2, \ldots, v_n = v'$ and $d \in \lambda(v) \cap \lambda(v')$;

. A user $v \in U$ is authorized for permission $v' \in P$ with respect to object $v'' \in O$ at point $d \in \mathcal{D}$ if and only if there exists an acs-path $v = v_1, v_2, \ldots, v_i, \ldots, v_{n-1} = v', v_n = v''$ such that $v_i \in R$ for some i, $v_1, \ldots, v_i$ is an act-path, $v_i, \ldots, v_{n-1}$ is an u-path, $(v_{n-1}, v_n) \in PO$ and and $d \in \lambda(v) \cap \lambda(v_i) \cap \lambda(v') \cap \lambda(v'')$;

4. SEPARATION OF DUTIES CONSTRAINTS

Separation of duties (SoD) prevents the occurrence of fraud arising out of conflicts of interests in organizations [Simon and Zurko 1997]. Separation of duties ensure that conflicting roles are not assigned to the same user or that conflicting permissions are not assigned to the same role.

Separation of Duty (SoD) comes in two varieties. First one is with respect to the mutual exclusion relation between a pair of roles. This is to guarantee that no user can be assigned to two conflicting roles. The second one is with respect to the mutual exclusion relation between two permissions. This is to guarantee that no role can be assigned two conflicting permissions. We denote these two types of SoD by using SD^R and SD^P edges, respectively. Since SoD is a symmetric relationship, the SD^R and SD^P edges are bi-directional.

We next define the separation of duties for the standard and weak models. The SoDs defined for the standard and weak models are expressed in terms of the graph $G = (V, E, \lambda)$, where $E = UA \cup PA \cup PO \cup RH_a \cup RH_u \cup SD^R \cup SD^P$ and $V = U \cup R \cup P \cup O$. For these cases, the SoD is similar to the SoD constraints in traditional RBAC. These are given below.

SoD Constraints for STARBACD$^-$ and STARBACD$^=$.

User-Role Assignment. if $(r, r') \in SD^R$ then there are no two edges (u, r) and (u, r') such that $\{(u, r), (u, r')\} \subset UA$

Permission-Role Assignment. if $(p, p') \in SD^P$ then there are no two u-paths of the form $r = v_1, v_2, \ldots, v_n = p$ and $r = v'_1, v'_2, \ldots,$
$v'_n = p'$

Similar to other associations, we defined the spatio-temporal constraint for the separation of duties with a function $\mu : E \rightarrow 2^{\mathcal{D}}$. For $e = (v, v') \in SD^R \cup SD^P$, $\mu(v, v')$ denotes the set of points in space-time at which the association between v and v' (in this case, the SoD) is enabled. In particular,

—if $(r, r') \in SD^R$, $\mu(r, r')$ denotes the set of points in space-time at which the role-role separation of duties constraint is valid;
—if $(p, p') \in SD^P$, $\mu(p, p')$ denotes the set of points in space-time at which the permission-permission separation of duties constraint is valid.

The strong model is defined over the labeled graph $G = (V, E, \lambda, \mu)$, where $E = UA \cup PA \cup PO \cup RH_a \cup RH_u \cup SD^R \cup SD^P$ and $V = U \cup R \cup P \cup O$. The strong model allows specification of weaker forms of SoD constraints than those supported by the traditional RBAC. Specifically, it allows one to specify the spatio-temporal points at which the SoD constraints are valid.

SoD Constraints for STARBACD$^+$.

. **User-Role Assignment:** if $(r, r') \in SD^R$ then there are no two edges (u, r) and (u, r'), corresponding to some user u, where $\mu(u, r) \cap \mu(u, r') \cap \mu(r, r') \neq \emptyset$

. **Permission-Role Assignment:** if $(p, p') \in SD^P$ then there are no two u-paths $r = v_1, v_2, \ldots, v_n = p$ and $r = v'_1, v'_2, \ldots, v'_n = p'$ where $\hat{\mu}(v_1, v_n) \cap \hat{\mu}(v'_1, v'_n) \cap \mu(p, p') \neq \emptyset$

5. DELEGATION IN STARBACD

Many situations require the temporary transfer or granting of access rights belonging to a user/role to another user/role in order to accomplish a given task. For example, a department chair may delegate his privilege to the assistant chair while he is traveling. The entity that transfers or grants his privileges temporarily to another entity is referred to as the delegator and 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). 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. In the graphical representation of STARBACD, we define a function $\nu : (U \cup R) \times (R \cup P) \rightarrow (U \cup R)$ that maps the delegation to the delegator. We assume the existence of different types of relationship corresponding to the different types of delegation as follows:

—User to User Role Delegation ($Delegate^R_{U2U}$) $= U \times R$, $\nu(u, r') = u'$ denotes the delegator who is a user authorized for role r'.
—User to User Permission Delegation ($Delegate^P_{U2U}$) $= U \times P$, $\nu(u, p') = u'$ denotes the delegator who is a user authorized for permission p'.
—User to Role Role Delegation ($Delegate^R_{U2R}$) $= R \times R$, $\nu(r, r') = u'$ denotes the delegator who is a user authorized for role r'.
—User to Role Permission Delegation ($Delegate^P_{U2R}$) $= R \times P$, $\nu(r, p') = u'$ denotes the delegator who is a user authorized for permission p'.
—Role to Role Role Delegation ($Delegate^R_{R2R}$) $= R \times R$, $\nu(r, r'') = r'$ denotes the delegator which is a role authorized for role r''. Note that r' and r'' can be the same role.
—Role to Role Permission Delegation ($Delegate^P_{R2R}$) $= R \times P$, $\nu(r, p') = r'$ denotes the delegator which is a role authorized for permission p'
—Role to User Role Delegation ($Delegate^R_{R2U}$) $= U \times R$, $\nu(u, r'') = r'$ denotes the delegator which is a role authorized for role r''. Note that r' and r'' can be the same role.
—Role to User Permission Delegation ($Delegate^P_{R2U}$) $= U \times P$, $\nu(u, p') = r'$ denotes the delegator which is a role authorized for permission p'.

Delegation in the Standard Model STARBACD$^=$

To represent delegation in a graph-theoretic manner for the standard and weak models, we have the labeled graph $G = (V, E, \lambda)$, where $E = UA \cup PA \cup PO \cup RH_a \cup RH_u \cup DG$, such that $DG = Delegate^R_{U2U} \cup Delegate^P_{U2U} \cup Delegate^R_{U2R} \cup Delegate^P_{U2R} \cup Delegate^R_{R2R} \cup Delegate^P_{R2R} \cup Delegate^R_{R2U} \cup Delegate^P_{R2U}$, and

$V = U \cup R \cup P \cup O$. We use the notations λ and $\hat{\lambda}$ as before. The constraints below describe the situations when delegation is possible in our spatio-temporal model. For instance, the first constraint gives the spatio-temporal constraints that must be satisfied when user u' wants to delegate role r' to another user u. It states that this delegation is possible only if there is some overlap among the set of spatio-temporal points associated with user u''s activation of role r' with those of user u's session creation. The other constraints are specified in a similar manner.

Delegation in STARBACD$^{=}$.

. If $(u, r') \in Delegate^{R}_{U2U}$ and $\nu(u, r') = u'$, then there exists an act-path $u' = v_1, v_2, \ldots, v_n = r'$ such that $\hat{\lambda}(v_1, v_n) \cap \lambda(u) \neq \emptyset$

. If $(u, p') \in Delegate^{P}_{U2U}$ and $\nu(u, p') = u'$, then there exists a path $u' = v_1, v_2, \ldots, v_i, \ldots, v_n = p'$ such that $v_i \in R$ for some i, $v_1, \ldots, v_i$ is an act-path, $v_{i+1}, \ldots, v_n$ is a u-path such that $\hat{\lambda}(v_1, v_n) \cap \lambda(u) \neq \emptyset$

. If $(r, r') \in Delegate^{R}_{U2R}$ and $\nu(r, r') = u'$, then there exists an act-path $u' = v_1, v_2, \ldots, v_n = r'$ such that $\hat{\lambda}(v_1, v_n) \cap \lambda(r) \neq \emptyset$

. If $(r, p') \in Delegate^{P}_{U2R}$ and $\nu(r, p') = u'$, then there exists a path $u' = v_1, v_2, \ldots, v_i, \ldots, v_n = p'$ such that $v_i \in R$ for some i, $v_1, \ldots, v_i$ is an act-path, $v_{i+1}, \ldots, v_n$ is a u-path such that
$\hat{\lambda}(v_1, v_n) \cap \lambda(r) \neq \emptyset$

. If $(r, r'') \in Delegate^{R}_{R2R}$ and $\nu(r, r'') = r'$, then there exists a path $r' = v_1, v_2, \ldots, v_n = r''$ where $(v_i, v_{i+1}) \in RH_a$ for $1 \leq i \leq (n-1)$ such that $\hat{\lambda}(v_1, v_n) \cap \lambda(r) \neq \emptyset$

. If $(r, p') \in Delegate^{P}_{R2R}$ and $\nu(r, p') = r'$, then there exists a u-path $r' = v_1, v_2, \ldots, v_n = p'$ such that $\hat{\lambda}(v_1, v_n) \cap \lambda(r) \neq \emptyset$

. If $(u, r'') \in Delegate^{R}_{R2U}$ and $\nu(u, r'') = r'$, then there exists a path $r' = v_1, v_2, \ldots, v_n = r''$ where $(v_i, v_{i+1}) \in RH_a$ for $1 \leq i \leq (n-1)$ such that $\hat{\lambda}(v_1, v_n) \cap \lambda(u) \neq \emptyset$

. If $(u, p') \in Delegate^{P}_{R2U}$ and $\nu(u, p') = r'$, then there exists a u-path $r' = v_1, v_2, \ldots, v_n = p'$ such that $\hat{\lambda}(v_1, v_n) \cap \lambda(u) \neq \emptyset$

Delegation in the Weak Model STARBACD^{-}

The weak model is defined on the same graph as the standard model given in Section 5. In the weak model, the entity v is authorized for the delegated entity v' if both entities v and v' are enabled. There is no requirement that the intermediate nodes on the path are enabled. The constraints shown below describe when delegation is possible in the weak model. The first constraint says that the user u' can delegate role r' to another user u if the user u' can activate the role r' in the weak model and provided the spatio-temporal points when user u can create a session and role r' can be activated have some overlap. The other constraints are defined in a similar manner.

Delegation in STARBACD^{-}.

. If $(u, r') \in Delegate^{R}_{U2U}$ and $\nu(u, r') = u'$, then there exists an act-path $u' = v_1, v_2, \ldots, v_n = r'$ such that $\lambda(v_1) \cap \lambda(v_n) \neq \emptyset$ and $\lambda(u) \cap \lambda(r') \neq \emptyset$

. If $(u, p') \in Delegate^{P}_{U2U}$ and $\nu(u, p') = u'$, then there exists a path $u' = v_1, v_2, \ldots, v_i, \ldots, v_n = p'$ such that $v_i \in R$ for some i, $v_1, \ldots, v_i$ is an act-path, $v_{i+1}, \ldots, v_n$ is an u-path such that $\lambda(v_1) \cap \lambda(v_n) \neq \emptyset$ and $\lambda(u) \cap \lambda(p') \neq \emptyset$

. If $(r, r') \in Delegate^{R}_{U2R}$ and $\nu(r, r') = u'$, then there exists an act-path $u' = v_1, v_2, \ldots, v_n = r'$ such that $\lambda(v_1) \cap \lambda(v_n) \neq \emptyset$ and $\lambda(r) \cap \lambda(r') \neq \emptyset$

. If $(r, p') \in Delegate^{P}_{U2R}$ and $\nu(r, p') = u'$, then there exists a path $u' = v_1, v_2, \ldots, v_i, \ldots, v_n = p'$ where $v_i \in R$ for some i, $v_1, \ldots, v_i$ is an act-path, $v_{i+1}, \ldots, v_n$ is an u-path such that $\lambda(v_1) \cap \lambda(v_n) \neq \emptyset$ and $\lambda(r) \cap \lambda(p') \neq \emptyset$

. If $(r, r'') \in Delegate^{R}_{R2R}$ and $\nu(r, r'') = r'$, then there exists a path $r' = v_1, v_2, \ldots, v_n = r''$ where $(v_i, v_{i+1}) \in RH_a$ for $1 \leq i \leq (n-1)$ such that $\lambda(v_1) \cap \lambda(v_n) \neq \emptyset$ and $\lambda(r) \cap \lambda(r'') \neq \emptyset$

. If $(r, p') \in Delegate^{P}_{R2R}$ and $\nu(r, p') = r'$, then there exists a u-path $r' = v_1, v_2, \ldots, v_n = p'$ such that $\lambda(v_1) \cap \lambda(v_n) \neq \emptyset$ and $\lambda(r) \cap \lambda(p') \neq \emptyset$

. If $(u, r'') \in Delegate^{R}_{R2U}$ and $\nu(u, r'') = r'$, then there exists a path $r' = v_1, v_2, \ldots, v_n = r''$ where $(v_i, v_{i+1}) \in RH_a$ for $1 \leq i \leq (n-1)$ such that $\lambda(v_1) \cap \lambda(v_n) \neq \emptyset$ and $\lambda(u) \cap \lambda(r'') \neq \emptyset$

. If $(u, p') \in Delegate^{P}_{R2U}$ and $\nu(u, p') = r'$, then there exists a u-path $r' = v_1, v_2, \ldots, v_n = p'$ such that $\lambda(v_1) \cap \lambda(v_n) \neq \emptyset$ and $\lambda(u) \cap \lambda(p') \neq \emptyset$

Delegation in the Strong Model STARBACD$^+$

The spatio-temporal constraints enforced in the delegation of the STARBACD$^+$ model are denoted with a function $\nu : E \rightarrow 2^{\mathcal{D}}$. For $e = (v, v') \in E$, $\mu(v, v')$ denotes the set of points in space-time at which the association between v and v' is activated.

The strong model is defined over the labeled graph $G = (V, E, \lambda, \mu)$, where $E = UA \cup PA \cup PO \cup RH_a \cup RH_u \cup DG$ and DG is the set of all delegation edges and $V = U \cup R \cup P \cup O$. The constraints for the strong model are enumerated below. The first constraint says that when a user u' delegates role r' to user u, then the delegation is possible only if the spatio-temporal points for activating user u''s role r' overlap with those in which the delegation is valid.

Delegation in STARBACD$^+$.

. If $(u, r') \in Delegate^{R}_{U2U}$ and $\nu(u, r') = u'$, then there exists an act-path $u' = v_1, v_2, \ldots, v_n = r'$ such that $\hat{\mu}(v_1, v_n) \cap \mu(u, r') \neq \emptyset$

. If $(u, p') \in Delegate^{P}_{U2U}$ and $\nu(u, p') = u'$, then there exists a path $u' = v_1, v_2, \ldots, v_i, \ldots, v_n = p'$ where $v_i \in R$ for some i, $v_1, \ldots, v_i$ is an act-path, $v_{i+1}, \ldots, v_n$ is an u-path such that $\hat{\mu}(v_1, v_n) \cap \mu(u, p') \neq \emptyset$

. If $(r, r') \in Delegate^{R}_{U2R}$ and $\nu(r, r') = u'$, then there exists an act-path $u' = v_1, v_2, \ldots, v_n = r'$ such that $\hat{\mu}(v_1, v_n) \cap \mu(r, r') \neq \emptyset$

. If $(r, p') \in Delegate^{P}_{U2R}$ and $\nu(r, p') = u'$, then there exists a path $u' = v_1, v_2, \ldots, v_i, \ldots, v_n = p'$ such that $v_i \in R$ for some i, $v_1, \ldots, v_i$ is an act-path, $v_{i+1}, \ldots, v_n$ is an u-path such that $\hat{\mu}(v_1, v_n) \cap \mu(r, p') \neq \emptyset$

. If $(r, r') \in Delegate^{R}_{R2R}$ and $\nu(r, r'') = r'$, then there exists a path $r' = v_1, v_2, \ldots, v_n = r''$ where $(v_i, v_{i+1}) \in RH_a$ for $1 \leq i \leq (n-1)$ such that $\hat{\mu}(v_1, v_n) \cap \mu(r, r') \neq \emptyset$

. If $(r, p') \in Delegate^{P}_{R2R}$ and $\nu(r, p') = r'$, then there exists an u-path $r' = v_1, v_2, \ldots, v_n = p'$ such that $\hat{\mu}(v_1, v_n) \cap \mu(r, p') \neq \emptyset$

. If $(u, r'') \in Delegate^{R}_{R2U}$ and $\nu(u, r'') = r'$, then there exists a path $r' = v_1, v_2, \ldots, v_n = r''$ where $(v_i, v_{i+1}) \in RH_a$ for $1 \leq i \leq (n-1)$ such that $\hat{\mu}(v_1, v_n) \cap \mu(u, r'') \neq \emptyset$

. If $(u, p') \in Delegate^{P}_{R2U}$ and $\nu(u, p') = r'$, then there exists an u-path $r' =$

$v_1, v_2, \ldots, v_n = p'$ such that $\hat{\mu}(v_1, v_n) \cap \mu(u, p') \neq \emptyset$

6. DYNAMIC MODEL

The pervasive computing applications are dynamic in nature–the accessing entities may change, resources requiring protection may be created or modified, and an entity's access to resources may change during the course of the application. Such changes may result in the unreachable entities or the violation of separation of duty constraints. We need to analyze the model to detect such problems. The following changes are possible in our model.

(1) **Entity and Relationship Removal** The following entities can be removed: user, role, permission, or object. Note that, this removal must be accompanied by deleting the relationships associated with these entities.
(2) **Relationship Removal** The following relationships can be removed: User-Role Assignment, Permission Usage Hierarchy, Role Activation Hierarchy, Role-Permission Assignment, or Permission-Object Assignment. This type of change can also cause an entity to become isolated.
(3) **Relationship Creation** A new relationship can be created between existing entities. The relationship may be User-Role Assignment, Permission Usage Hierarchy, Role Activation Hierarchy, Role-Permission Assignment, Permission-Object Assignment, SoD, or Delegation. Creation of a new relationship may result in separation of duty violation.
(4) **Entity and Relationship Creation** A new entity together with its corresponding new relationship can be created. The entity may be user, role, permission, or object. The relationship may be User-Role Assignment, Permission Usage Hierarchy, Role Activation Hierarchy, Role-Permission Assignment, Permission-Object Assignment, SoD, or Delegation depending on the type of entity being created. This type of change can cause the SoD constraints violation.
(5) **Updating Spatio-Temporal Constraints** The spatio-temporal constraints assigned to entities or relations can be changed. The entity may be user, role, permission, or object. The relationship may be User-Role Assignment, Permission Usage Hierarchy, Role Activation Hierarchy, Role-Permission Assignment, Permission-Object Assignment, SoD, or Delegation. This type of change can cause either the infeasible path violation or SoD constraints violation.

6.1 Algorithm for Detecting Isolated Entities

6.1.1 *Preliminaries.* We define an isolated entity as one which is unreachable and therefore cannot be used. The isolated entity can be determined by considering the *in-degree* and *out-degree* of each vertex. The *in-degree* of the vertex are defined with respect to the type of STARBACD model.

STARBACD$^{=}$ *and STARBACD*$^{-}$. In the labeled graph $G = (V, E, \lambda)$, where $V = U \cup R \cup P \cup O$ and $E = UA \cup PA \cup PO \cup RH_a \cup RH_u$, *in-degree* of a vertex v is the cardinality of the set $\{(v', v) | ((v', v) \in E) \wedge (\lambda(v') \cap \lambda(v) \neq \emptyset)\}$

STARBACD^{+}. In the labeled graph $G = (V, E, \lambda, \mu)$, where $V = U \cup R \cup P \cup O$ and $E = UA \cup PA \cup PO \cup RH_a \cup RH_u$, *in-degree* of a vertex v is the cardinality of the set $\{(v', v) | ((v', v) \in E) \wedge (\lambda(v') \cap \lambda(v) \cap \mu(v', v) \neq \emptyset)\}$

Source is a vertex v which *in-degree*$(v) = 0$
Similarly, we define the *out-degree* of the vertex as follow:

STARBACD$^{=}$ *and STARBACD*$^{-}$. In the labeled graph $G = (V, E, \lambda)$, where $V = U \cup R \cup P \cup O$ and $E = UA \cup PA \cup PO \cup RH_a \cup RH_u$, *out-degree* of a vertex v is the cardinality of the set $\{(v, v')|((v, v') \in E) \wedge (\lambda(v) \cap \lambda(v') \neq \emptyset)\}$
STARBACD^{+}. In the labeled graph $G = (V, E, \lambda, \mu)$, where $V = U \cup R \cup P \cup O$ and $E = UA \cup PA \cup PO \cup RH_a \cup RH_u$, *out-degree* of a vertex v is the cardinality of the set $\{(v, v')|((v, v') \in E) \wedge (\lambda(v) \cap \lambda(v') \cap \mu(v, v') \neq \emptyset)\}$

Sink is a vertex v which *out-degree*$(v) = 0$
Note that, we do not consider separation of duty or the delegation edges since the modifications to these edges do not change the isolated entities.

6.1.2 *The Detection Algorithm.* The different types of isolated entities are detected as follows:

User. For $v \in U$, v is the isolated entity iff *out-degree*$(v) = 0$
Role and Permission. For $v \in R \cup P$, v is the isolated entity iff (*in-degree*$(v) = 0$) $\vee$ (*out-degree*$(v) = 0$)
Object. For $v \in O$, v is the isolated entity iff *in-degree*$(v) = 0$

To get the *in-degree* and *out-degree*, we have to count the number of edges connected to each vertex. This can be done in $O(VE)$ time. However, we can improve this by recording the *in-degree* and *out-degree* of each vertex. Each time the vertex or the edge is added to or removed from the graph, we update the *in-degree* and *out-degree* of the related vertices. Since we do not allow the existence of multiple edges between each pair of vertices, this update process can be done in $O(V)$. After we have such values recorded for every vertex, the detection can be done in $O(V)$.

6.2 Algorithm for Detecting Infeasible Paths

6.2.1 *Preliminaries.* In STARBACD model, a user u is authorized for permission p through role r with respect to object o iff there exists a valid *acs-path* which contains u, r, p, and o. We define an infeasible path as an invalid *acs-path* i.e. an *acs-path* which cannot grant the authorization of any permission to user.

6.2.2 *The Detection Algorithm.* To detect the infeasible path, we assume that we store all source vertices in a list. Each member in the list maintain its own depth-first search (DFS) tree. To generate these trees, we perform DFS from each source. While performing the DFS, we check if there is any spatio-temporal conflicts between the nodes (for STARBACD$^{=}$ and STARBACD^{-}) or edges (for STARBACD^{+}). If there is any conflict, then there exists an infeasible path. This step could be done in $O(VE)$. After the process we will have set of the initial DFS trees which consists of feasible paths. Next for each update operation of the graph, we ensure that the following conditions are satisfied:

—Only *user* vertices can be the root of each subtree.
—Only *object* vertices can be the leaf node of each subtree.

For each update operation of the graph, we perform the following:
If any new entity v and its corresponding relationship have been added to the initial graph, we consider the following:

—If v is a new source, we perform DFS from v to create all of its *acs-path*s. While performing the DFS, we check whether the spatio-temporal constraints between the source and its successors are satisfied. If so, we add v to the source list and maintain its pointers to its immediate successors. If not, then this v will create an infeasible path. This step can be done in $O(E)$ time.
—If v is a new intermediate vertex, we perform DFS from each source. While performing the DFS, we check whether all spatio-temporal constraints are satisfied. If so, we create pointer from v's immediate predecessors to v, and from v to its immediate successors. If not, then this v will create an infeasible path. This step can be done in $O(VE)$ time.
—If v is a new sink, we perform reverse DFS from v. While performing the reverse DFS, we check whether the spatio-temporal constraints between v and its predecessors are satisfied. If so, we create pointer from its immediate predecessors to v. If not, then this v will create an infeasible path. This step can be done in $O(E)$ time.

If any existing spatio-temporal constraint has been updated in the initial graph, we consider the following:

—If the update is done on $\lambda(v)$, where v is a source, we perform DFS from v to each of its *acs-path*. While performing the DFS, we check whether the spatio-temporal constraints between the source and its successors are satisfied. If so, we update $\lambda(v)$ to the new one. If not, then this update will create an infeasible path. This step can be done in $O(E)$ time.
—If the update is done on $\mu(v, v')$, where v is a source, we perform DFS from v to each of its $acs - path$ which contains v'. While performing the DFS, we check whether the spatio-temporal constraints between the source and its successors are satisfied. If so, we update $\mu(v, v')$ to the new one. If not, then this update will create an infeasible path. This step can be done in $O(E)$ time.
—If the update is done on $\lambda(v)$ or $\mu(v, v')$, where v is an intermediate vertex, we perform DFS from each source. While performing the DFS, we check whether all spatio-temporal constraints are satisfied. If so, we update $\lambda(v)$ or $\mu(v, v')$ to the new one. If not, then this update will create an infeasible path. This step can be done in $O(VE)$ time.
—If the update is done on $\lambda(v)$, where v is a sink, we perform reverse DFS from v. While performing the reverse DFS, we check whether the spatio-temporal constraints between v and its predecessors are satisfied. If so, we update $\lambda(v)$ to the new one. If not, then this v will create an infeasible path. This step can be done in $O(E)$ time.
—If the update is done on $\mu(v, v')$, where v' is a sink, we perform DFS from v' to each of its *acs-path* which contains v. While performing the DFS, we check whether the spatio-temporal constraints between the source and its successors are satisfied. If so, we update $\mu(v, v')$ to the new one. If not, then this update will create an infeasible path. This step can be done in $O(E)$ time.

6.3 Algorithm for Detecting SoD Violations

6.3.1 *Preliminaries.* In STARBACD model, the SoD can be violated by two ways. First, if $(r_1, r_2) \in SD^R$, and there exists *acs-paths* from u_1 to r_1 and u_1 to

r_2. Or, if $(p_1, p_2) \in SD^P$, and there exists *u-paths* from r_1 to p_1 and r_1 to p_2.

6.3.2 *The Detection Algorithm.* Consider the dynamic case where edges can be added and deleted from the graph. The naive algorithm can be done by performing the reverse DFS on each $(v, v') \in SD^R \cup SD^P$ of the modified graph to find the common predecessor. This could be done in $O(k|E|)$ time. We can apply the same algorithm for the case where the spatio-temporal constraint is updated in the graph too.

Our algorithm which will be proposed next is the special case of the algorithm to find the common predecessors in a Directed Acyclic Graph (DAG) described in detail in the Appendix. In our algorithm, each entity except users will maintain a list of users authorized for it by performing the DFS from each user. Only users satisfying the spatio-temporal constraints will be added to the list. To determine whether the SoD $(v, v') \in SD^P \cup SD^R$ is violated, we compare whether $u \in U$ is in the authorized users list of both v and v', and $\lambda(u) \cap \mu(v, v') \neq \emptyset$. If this evaluates to true, then there exists an SoD violation. Since the size of each list cannot exceed the number of user vertices, the evaluation time is $O(|U|)$. Let k be number of SoD edges, the detection time for the static case where no adding or removing edges allow is equal to $O(k|U|)$. To label all vertices takes $O(|E||U|)$ time, yields the total running time in the static graph equal to $O((k+|E|)|U|)$. However, in case that all edges modifications are of same type i.e. only either adding edges or deleting edges are allowed, we can improve the running time by applying the following graph specification updating summarized below:

—When only adding edges is allowed, each time that new edge is added, we update only the label list of vertices belonged to the graph portion that have not been reached before by using the Incremental-DFS described in the Appendix. All updates take $O(|E||U|)$ time, and detecting whether the SoD is violated take $O(|U|)$ per SoD edge. This yields the total processing time equal to $O((k + |E|)|U|)$.

—When only removing edges is allowed, we update only the label list of vertices that becomes unreachable by some user u after the edge removal. Following the algorithm described in the Appendix, the removal of an edge takes $O(|E|log|V|)$ time for relabeling for each user vertex, and detecting whether the SoD is violated take $O(|U|)$ per SoD edge. This yields the total processing time equal to $O((k + |E|log|V|)|U|)$.

For the detail on graph specification updating algorithm and proof of correctness, we refer to the Appendix.

7. EXAMPLE SCENARIO

In this example, we describe a military application where the STARBACD$^+$ can be applied. Let us assume that in the battlefield, each troop consists of military staff with the following responsibilities: The *Intelligent Officer* is responsible for the process of acquiring enemy information, interpreting it and then sending it to the *Soldier* in his troop. The *Clinical Officer* is in charge of monitoring the health information of his troop, evaluating the information to check whether the trooper's life is in danger, and sending the SOS signal to the commander to get the proper

help. The list of entities and the spatio-temporal relationships are shown in Tables I and II respectively.

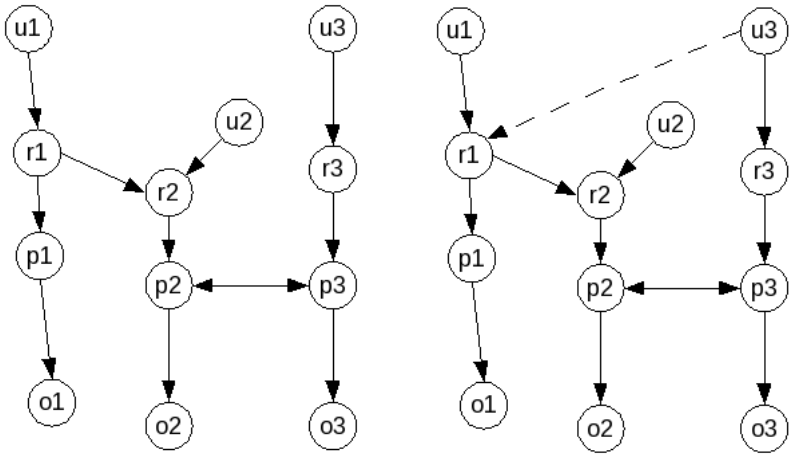
Name	Description	Spatio-Temporal Domain (λ)
u_1	Alex	[*Universe*, *Always*]
u_2	Ben	[*Universe*, *Always*]
u_3	Charlie	[*Universe*, *Always*]
r_1	Intelligence Officer	[*Universe*, *Always*]
r_2	Soldier	[*Field*, *Always*]
r_3	Clinical Officer	[*Universe*, *Always*]
p_1	Access Surveillance Sensor	[*Universe*, *Always*]
p_2	Maneuver the Vehicle	[*Field*, *Always*]
p_3	Access Vital Sensor	[*Universe*, *Always*]
o_1	Surveillance Sensor Information	[*Universe*, *Always*]
o_2	Tank	[*Field*, *Always*]
o_3	Health Information	[*Universe*, *Always*]

Table I. STARBACD Entities for the Example

Name	Description	Spatio-Temporal Domain (μ)
(u_1, r_1)	User-Role Assignment	[*Universe*, *Always*]
(u_2, r_2)	User-Role Assignment	[*Field*, *Always*]
(u_3, r_3)	User-Role Assignment	[*Universe*, *Always*]
(r_1, r_2)	Permission Usage Hierarchy	[*Field*, *Always*]
(r_1, p_1)	Permission-Role Assignment	[*Universe*, *Always*]
(r_2, p_2)	Permission-Role Assignment	[*Field*, *Always*]
(r_3, p_3)	Permission-Role Assignment	[*Universe*, *Always*]
(p_2, p_3)	Separation of Duties	[*Universe*, *Always*]
(p_3, p_2)	Separation of Duties	[*Universe*, *Always*]
(p_1, o_1)	Permission-Object Assignment	[*Universe*, *Always*]
(p_2, o_2)	Permission-Object Assignment	[*Field*, *Always*]
(p_3, o_3)	Permission-Object Assignment	[*Universe*, *Always*]

Table II. STARBACD Relationships and Constraints

The graph-theoretic representation is shown in Figure 1(a). We will only describe parts of this configuration. User *Alex* (u_1) can create session at any time and at any place as per Row 1 of Table I. He is assigned the role of *Intelligence Officer* (r_1) which can be activated at any place at any time. During this time and at this location, he has permission (p_1) to access the *Surveillance Sensor Information* (o_1). Since *Intelligence Officer* is senior to *Soldier* role in the permission usage hierarchy, he can also get the permission to maneuver the *Tank*. However, this permission is allowed only when the hierarchy is enabled on the battlefield. During the war, Alex gets injured and cannot pursue his mission. So, his role must be delegated to Charlie until he fully recovers. This new graphical representation is shown in Figure 1(b) where the delegation edge is represented by the dashed arrow. However, this delegation should not be allowed because our algorithm detects a violation of separation of duty constraint in the presence of this delegation.



(a) Configuration before delegation (b) Configuration after delegation

Fig. 1. STARBACD Configuration for Example

8. CONCLUSION AND FUTURE WORK

Traditional access control models do not take into account environmental factors before making access decisions and may not be suitable for pervasive computing applications. Towards this end, we proposed a spatio-temporal role based access control model that supports delegation. We identified the entities and relations in RBAC and investigated their dependence on location and time. The different types of role hierarchy, separation of duty, and delegation that can be supported by constraining the location and time parameters are described in this paper. The dependence on location and time necessitates changes in the invariants and the operations of RBAC. The behavior of the model is formalized using graph-theoretic notations. This not only allows us to visualize the model but also helps in studying the interactions of the different constraints in the model.

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. Once we have an implementation, we can validate our model using some real-world application. We also plan to extend our model for supporting workflow applications. Workflow applications have their own set of constraints. It will be interesting to investigate the interactions of workflow application constraints and authorization constraints.

ACKNOWLEDGMENTS

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

REFERENCES

Ardagna, C. A., Cremonini, M., Damiani, E., di Vimercati, S. D. C., and Samarati, P. 2006. Supporting location-based conditions in access control policies. In *Proceedings of the ACM Symposium on Information, Computer and Communications Security*. Taipei, Taiwan, 212–222.

Atluri, V. and Chun, S. A. 2004. An authorization model for geospatial data. *IEEE Transactions on Dependable and Secure Computing 1,* 4 (October-December), 238–254.

Atluri, V. and Chun, S. A. 2007. A geotemporal role-based authorisation system. *International Journal of Information and Computer Security 1,* 1/2 (January), 143–168.

Barka, E. and Sandhu, R. 2000a. A Role-Based Delegation Model and Some Extensions. In *Proceeding of the 23rd National Information Systems Security Conference*. Baltimore, MD, USA.

Barka, E. and Sandhu, R. 2000b. Framework for role-based delegation models. In *Proceedings of the 16th Annual Computer Security Applications Conference*. New Orleans, LA, USA, 168–176.

Barka, E. and Sandhu, R. 2004. Role-Based Delegation Model/ Hierarchical Roles (RBDM1). In *Proceedings of the 20th Annual Computer Security Applications Conference*. Los Alamitos, CA, USA, 396–404.

Bertino, E., Bonatti, P. A., and Ferrari, E. 2000. TRBAC: a temporal role-based access control model. In *Proceedings of the 5th ACM Workshop on Role-Based Access Control*. Berlin, Germany, 21–30.

Bertino, E., Catania, B., Damiani, M. L., and Perlasca, P. 2005. GEO-RBAC: a spatially aware RBAC. In *Proceedings of the 10th ACM Symposium on Access Control Models and Technologies*. Stockholm, Sweden, 29–37.

Chandran, S. M. and Joshi, J. B. D. 2005. *LoT-RBAC*: A Location and Time-Based RBAC Model. In *Proceedings of the 6th International Conference on Web Information Systems Engineering*. New York, NY, USA, 361–375.

Chen, L. and Crampton, J. 2008. On Spatio-Temporal Constraints and Inheritance in Role-Based Access Control. In *Proceedings of the 2008 ACM Symposium on Information, Computer and Communications Security*. Tokyo, Japan, 205–216.

Covington, M. J., Fogla, P., Zhan, Z., and Ahamad, M. 2002. A Context-Aware Security Architecture for Emerging Applications. In *Proceedings of the Annual Computer Security Applications Conference*. Las Vegas, NV, USA, 249–260.

Covington, M. J., Long, W., Srinivasan, S., Dey, A., Ahamad, M., and Abowd, G. 2001. Securing Context-Aware Applications Using Environment Roles. In *Proceedings of the 6th ACM Symposium on Access Control Models and Technologies*. Chantilly, VA, USA, 10–20.

Crampton, J. and Khambhammettu, H. 2008. Delegation in role-based access control. *International Journal of Information Security 7,* 2 (March), 123–136.

Ferraiolo, D. F., Sandhu, R. S., Gavrila, S. I., Kuhn, D. R., and Chandramouli, R. 2001. Proposed NIST Standard for Role-Based Access Control. *ACM Transactions on Information and Systems Security 4,* 3 (August), 224 – 274.

Franciosa, P. G., Gambosi, G., and Nanni, U. 1997. The Incremental Maintenance of a Depth-First-Search Tree in Directed Acyclic Graphs. *Information Processing Letters 61,* 2 (January), 113–120.

Hengartner, U. and Steenkiste, P. 2004. Implementing Access Control to People Location Information. In *Proceedings of the 9th ACM Symposium on Access Control Models and Technologies*. Yorktown Heights, NY, USA, 11–20.

Hulsebosch, R. J., Salden, A. H., Bargh, M. S., Ebben, P. W. G., and Reitsma, J. 2005. Context sensitive access control. In *Proceedings of the 10th ACM Symposium on Access Control Models and Technologies*. New York, NY, USA, 111–119.

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

Joshi, J. B. D. and Bertino, E. 2006. Fine-grained role-based delegation in presence of the hybrid role hierarchy. In *Proceedings of the 11th ACM Symposium on Access Control Models and Technologies*. Lake Tahoe, California, USA, 81–90.

Joshi, J. B. D., Bertino, E., Ghafoor, A., and Zhang, Y. 2008. Formal foundations for hybrid hierarchies in GTRBAC. *ACM Transactions on Information and System Security 10,* 4 (January), 1–39.

Leonhardt, U. and Magee, J. 1997. Security Consideration for a Distributed Location Service. *Imperial College of Science, Technology and Medicine, London, UK*.

Nyanchama, M. and Osborn, S. 1999. The role graph model and conflict of interest. *ACM Transactions on Information and System Security 2,* 1, 3–33.

Pu, F., Sun, D., Cao, Q., Cai, H., and Yang, F. 2006. Pervasive Computing Context Access Control Based on $UCON_{ABC}$ Model. In *Proceedings of the 2nd International Conference on Intelligent Information Hiding and Multimedia Signal Processing*. Pasadena, CA, 689–692.

Ray, I. and Kumar, M. 2006. Towards a location-based mandatory access control model. *Computers & Security 25,* 1 (February), 36–44.

Ray, I., Kumar, M., and Yu, L. 2006. LRBAC: A Location-Aware Role-Based Access Control Model. In *Proceedings of the 2nd International Conference on Information Systems Security*. Kolkata, India, 147–161.

Ray, I., Li, N., France, R., and Kim, D.-K. 2004. Using UML to Visualize Role-Based Access Control Constraints. In *Proceedings of the 9th ACM symposium on Access Control Models and Technologies*. Yorktown Heights, NY, USA, 115–124.

Ray, I. and Toahchoodee, M. 2007. 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*. Redondo Beach, CA, 211–226.

Ray, I. and Toahchoodee, M. 2008. A Spatio-Temporal Access Control Model Supporting Delegation for Pervasive Computing Applications. In *Proceedings of the 5th International Conference on Trust, Privacy & Security in Digital Business*. Turin, Italy, 48–58.

Sampemane, G., Naldurg, P., and Campbell, R. H. 2002. Access Control for Active Spaces. In *Proceedings of the Annual Computer Security Applications Conference*. Las Vegas, NV, USA, 343–352.

Samuel, A., Ghafoor, A., and Bertino, E. 2007. A Framework for Specification and Verification of Generalized Spatio-Temporal Role Based Access Control Model. Tech. rep., Purdue University. February. CERIAS TR 2007-08.

Simon, R. and Zurko, M. E. 1997. Separation of Duty in Role-based Environments. In *Proceedings of the 10th Computer Security Foundations Workshop*. Rockport, MA, USA, 183–194.

Toahchoodee, M. and Ray, I. 2008. On the Formal Analysis of a Spatio-Temporal Role-Based Access Control Model. In *Proceedings of the 22nd Annual IFIP WG 11.3 Working Conference on Data and Applications Security*. London, U.K., 17–32.

Toahchoodee, M. and Ray, I. 2009. Using Alloy to Analyze a Spatio-Temporal Access Control Model Supporting Delegation. *IET Information Security 3,* 3 (September), 75–113.

Yu, H. and Lim, E.-P. 2004. LTAM: A Location-Temporal Authorization Model. In *Secure Data Management*. Lecture Notes in Computer Science, vol. 3178. Toronto, Canada, 172–186.

Yuan, C., He, Y., He, J., and Zhou, Z. 2006. A Verifiable Formal Specification for RBAC Model with Constraints of Separation of Duty. In *Proceedings of the 2nd SKLOIS Conference on Information Security and Cryptology*. Beijing, China, 196–210.

Zhang, X., Oh, S., and Sandhu, R. 2003. PBDM: a flexible delegation model in RBAC. In *Proceedings of the 8th ACM Symposium on Access Control Models and Technologies*. Como, Italy, 149–157.

Appendix

Finding common predecessors in a DAG

Let $V = \{1, 2, \ldots, n\}$. Given a subset S of V, the *characteristic vector* of S is a bit vector that has a 1 in position i if and only if $i \in S$. Representing a subset's characteristic vector with a bit array allows one to determine whether $i \in S$ in $O(1)$ time.

In a directed graph $G = (V, E)$, let the *in-degree* of a vertex v be the number of edges directed into v. That is, the in-degree of v is the cardinality of the set $\{(u, v)|(u, v) \in E\}$. Similarly, the out-degree of v is the number of edges directed out of v, that is, the cardinality of the set $\{(v, u)|(v, u) \in E\}$.

An undirected graph is a special case of a directed graph where, for every directed edge (u, v), (v, u) is also a directed edge. In this case, we denote the pair $\{(u, v), (v, u)\}$ by uv. The *underlying undirected graph* of a directed graph is the graph obtained by adding (v, u) as an edge whenever (u, v) is an edge.

In a directed graph, and *in-neighbor* is a vertex u that has a directed edge (u, v) to v, and an *out-neighbor* is a vertex w such that v has a directed edge (v, w) to w. If G is a DAG, let a *predecessor* of vertex v be any vertex w such that there is a directed path from w to v. Similarly, a *successor* of v is any vertex u such that there is a directed path from v to u. A vertex is a *source* in a DAG if its in-degree is 0 and a *sink* if its out-degree is 0. A *path tree rooted at vertex* w is a subset of the edges of G that form a tree rooted at w, oriented away from w, and reaching every successor of w. (A DFS or BFS tree is a special case of a path tree.)

We consider variants of the following problem: Given a DAG $G = (V, E)$ and a pair $\{u, v\}$ of vertices, determine whether u and v have a common predecessor.

Let us call $\{u, v\}$ a *query*. Let $n = |V|$ be the number of vertices, $m = |E|$ be the number of edges, k be the number of queries. Let p denote the number of sources of G. We may assume that every vertex has either in-degree or out-degree greater than zero, since vertices failing this property are irrelevant to the problem and can be removed from the graph in linear time. Therefore $n = O(m)$, and a time bound of $O(n + m)$ can be simplified to $O(m)$.

Let G^T denote the *transpose* of G, which is obtained by reversing the directions of all edges of G. That is, $G^T = (V, E')$, where $E' = \{(v, u)|(u, v) \in E\}$. Given an adjacency-list representation of G, it is well known that it takes $O(n + m)$ time to find the adjacency lists of the transpose G^T by radix sorting the edges using source vertex as the secondary sort key and destination vertex as the primary sort key. An adjacency-list representation of a graph gives, for each vertex, a list of out-neighbors; this gives, for each vertex, a list of in-neighbors.

A naive algorithm for the static and dynamic cases

If the number k of queries is 1, the query can be answered in $O(m)$ time by performing depth-first search from u in G^T, marking all visited vertices, and then performing depth-first search from v in G^T, determining whether any marked vertices are encountered.

A sequence of k queries on k graphs, each with $O(m)$ edges, takes $O(km)$ time. This gives a time bound of $O(km)$ for the *dynamic* case where edges can be added to or deleted from G between queries, and m is the maximum number of edges the

graph has at any point.

Some improvements when k is large

Note that k can be quadratic in the number of vertices. If G is dynamic, then queries may be repeated as G changes, and there is no upper bound on k. We consider the possibility of better bounds than $O(km)$ in these cases.

In the static case, we observe that two vertices have a common predecessor if and only if they have a common predecessor that is a source. For each source w, we may label all successors of w by depth-first search. This gives each vertex at most p labels. Moreover, if a source is added to a vertex's list of labels, we add it to the back of the list. That way, all lists of labels are sorted in the order in which the sources were processed. This labeling takes $O(pm)$ time, and a query now takes $O(p)$ time to determine whether the two query vertices share a common label. Summarizing, this gives an $O((k+m)p)$ algorithm to add the labels and then process the k queries.

If $k = o(p)$, this is $O(mp)$, which is worse than the $O(km)$ bound we got above. If $p = o(k)$ and $k = O(m)$, the bound is still $O(mp)$, but this is asymptotically better than the $O(km)$ bound we got above, and if $m = o(k)$, then the bound is $O(kp)$ which is also better than $O(km)$, since $p = O(n) = O(m)$. Summarizing, this approach gives a better asymptotic bound when $p = o(k)$.

Adding edges to G between queries

Let us now consider how we might do better than $O(km)$ for the dynamic case if k is large and edges may be added, but not deleted. In this case, we may maintain a path tree rooted at each source w. Below, we see that we maintain the invariant that the path tree is a DFS tree. Vertices once again carry w in a sorted list of source labels if they are a successor of w, that is, if they are in w's path tree.

Initially, we compute a path tree from each source using DFS. Let (u, v) be an added edge. For each vertex w in u's list of labels, we extend the DFS tree rooted at w by performing a depth-first search from v, retreating whenever a vertex labeled w is encountered. Let us call this an *incremental DFS*. Using $O(p)$ space to store the characteristic bit vector of the set of labels at each vertex allows us to look up whether w is a label of a vertex in $O(1)$ time, using a total of $O(m + np)$ space. If v was previously a source, we may remove the tree rooted at v, since it is no longer a source. It is trivial to do this in $O(n) = O(m)$ time.

Each edge is traversed once over all incremental DFS's on w, giving a bound of $O(m)$ to update the path tree rooted at w over all edge insertions. The addition of an edge never creates a new source. This therefore gives an $O(mp)$ bound for all updates to trees, where p is the initial number of sources.

Determining whether two vertices lie in a common DFS tree once again takes $O(p)$ time, for a total of $O((k + m)p)$, where m is the final number of edges in G. The analysis of when this is better than the $O(km)$ bound we obtained above is the same as it is for the $O((k + m)p)$ bound we got for the static case.

In fact, it is possible to implement this algorithm without recording the trees, and only making use of the labels to guide the incremental DFS operations. However, if the trees are maintained, an interesting observation is that this maintains the invariant that each tree is a DFS tree. Suppose this is true for w's tree T_w before

it was extended to T'_w due to the addition of (u, v), giving a new graph G'. It is easy to see that in a DFS of G' where (u, v) is the last edge considered at u, T_w is the state of the depth-first tree on G' during a run of DFS just before (u, v) is considered, where (u, v) is the last edge in u's adjacency list.

Deleting edges from G between queries

Let us now suppose that edges are only deleted from G. For this, we maintain, for each source, a path tree rooted at the source. Once again, each vertex is labeled with a sorted list of sources that it is a successor of.

When a new source is created by the removal of an edge, we use DFS to get an initial path tree for the new source.

It remains to describe how to update an existing path tree T_w for a single source w after deletion of an edge; path trees for all sources are updated with the same procedure.

When an edge (u, v) is removed from G, we find whether (u, v) is an edge of T_w. If it is not, T_w remains a path tree, and we are done.

Definition 1. Let w be a source, let T_w be the current path tree rooted at w, and let (u, v) be an edge of G that is also an edge in T_w and that is deleted from G, yielding G'. Removal of (u, v) splits T_w into two subtrees, the subtree T_v of T_w rooted at v, and the remainder T'_w of T_w. The *status* of a vertex of G' is whether it is reachable from w in G'.

Before the deletion of (u, v) the status of all vertices of G is known: the ones that are reachable from w are just the vertices in T_w, and the remaining vertices are not reachable.

LEMMA 2. *Deletion of* (u, v) *can only change the status of some vertices in* T_v *from* reachable *to* unreachable. *If a vertex* x *of* T_v *continues to be reachable, then so does every vertex in the subtree* T_x *of* T_v *rooted at* x.

PROOF. Removal of an edge cannot make a vertex reachable from w if it was not reachable before, so the status of vertices not in T_w does not change. The status of vertices in T'_w does not change, since the edges of this tree give paths in G' from w to every vertex in T'_w. Let y be a vertex in x's subtree of T_v. If x continues to be reachable, then there is a path P of G' from w to x. Appending the unique tree path from x to y in T_x to P yields a directed path from w to y, which implies that y continues to be reachable. □

The goal is to determine the new status of each node x in T_v. Our strategy is to process the vertices of unknown status in an order such that when it is time to make the status of a vertex x known, the status of all in-neighbors of x is known. This reduces the problem of determining x's status to that of determining whether it has an in-neighbor that is known to be a successor of w.

LEMMA 3. *Let* $(v_1, v_2, \ldots v_n)$ *be a topological sort of* G. *If the status of vertices of unknown status is made known in the order in which they appear in this sort, then when it is time to make the status of a vertex* v_i *known, the status of all in-neighbors is known.*

PROOF. By induction on the number of vertices whose status is made known, when it is time to make v_i's status known, the status of all earlier vertices in topological order is known. All in-neighbors of a vertex are earlier in topological order. □

Let us give an overview of our strategy. Before beginning any operations on our initial DAG, we assign topological sort numbers to the vertices. Deletion of an edge does not invalidate a topological sort, so this numbering remains a valid topological sort after any number of edge deletions. Our strategy for obtaining our time bound is to take advantage of Lemma 3 by using a priority queue, keyed on topological-sort numbers, to dispense vertices of T_v in topological order. By Lemma 3, when a vertex x is dispensed from the priority queue, the reachability status of all in-neighbors is known. We determine whether x is reachable from w by determining whether it has an in-neighbor that is reachable from w. Moreover, when an in-neighbor is found to be unreachable, no subsequent edge deletion will make it reachable, so after each edge deletion, if a vertex x is inserted and dispensed from the priority queue, we may resume the search of its in-neighbor list where we left off the last time x was inserted and dispensed from the priority queue. This ensures that over all edge deletions, each element of x's in-neighbor list is examined only once to determine whether it is reachable.

A critical element for our time bound is to observe the following constraint on which vertices we can touch:

—**Constraint:** We touch a vertex x of T_v only if it becomes unreachable or has no reachable parent in T_v.

We accomplish this by inserting x to the priority queue only if $x = v$ or the status of its parent in T_v is found to be unreachable from w. If a vertex x is determined to have a reachable in-neighbor z, then, by Lemma 2, all vertices in T_x are reachable from w, so we can include all of them in the new path tree rooted at T_w in $O(1)$ time by adding (z, x) to the tree. This observes the constraint by avoiding touching lower vertices in T_x. If x is found not to be reachable, then we can touch the children of x in T_v. We insert these children in the priority queue.

LEMMA 4. *Vertices are dispensed from the priority queue in topological order.*

PROOF. When x is extracted, it has an earlier topological number of any vertex in the priority queue. If it is determined to be a successor of w, no new vertices are inserted before another extraction. If it is determined not to be a successor of w, its children in T_v are inserted, and since there is an edge of G from x to each of these children, they have larger topological numbers than x does. In either case, the minimum topological number in the priority queue increases every time a vertex is extracted. □

We can now give the detailed implementation that gives the time bound. A given vertex x might be inserted to the priority queue any time an edge is deleted. After the first time x is inserted to the priority queue, we maintain a pointer x_w into x's in-neighbor list. The pointer initially points to the beginning of x's in-neighbor list, and satisfies the following invariants:

—All elements of x's in-neighbor list that precede x_w are known to be non-successors of w or have ceased to be in-neighbors of x.
—After T_w is updated, if x is a node of T_w, then x_w points to its parent in T_w.

Whenever x is inserted in the priority queue, it has lost its parent in T_w or the parent has ceased to be reachable from w. In either case, we can advance x_w without violating the first constraint. We iteratively advance x_w until we find an in-neighbor z that is known to be a successor of of w, or reach the end of the in-neighbor list. If z is found, we leave x_w pointing to z, and make z x's parent, satisfying the second invariant. If it is not found, we label x as unreachable and insert its children in the priority queue.

LEMMA 5. *The foregoing algorithm correctly updates the status of all nodes as reachable or not reachable from w after an edge deletion, and modifies T_w to be a correct path tree.*

PROOF. That the invariants are maintained on x_w follows from the fact that once an in-neighbor is labeled as unreachable, it is never relabeled as reachable, since edge insertions are not allowed. That the status of the in-neighbors of x are all correctly labeled whenever x is extracted from the priority queue follows from Lemmas 3 and 4. It follows that x is correctly labeled. If x remains reachable, that the descendants of x remain correctly labeled and included in a correct path tree follows from Lemma 2. Since x is an arbitrary node of T_v that is inserted to and extracted from the priority queue, and all nodes of T_v are either inserted and extracted from the priority queue, it follows that all vertices of T_v are either correctly labeled as non-successors of w, or are correctly linked into a new path tree rooted at w. □

LEMMA 6. *The above algorithm takes $O(m \log n)$ time over all edge deletions.*

PROOF. Every time a node x is inserted to the priority queue, x_w is advanced in its adjacency list. The time spent over all insertions of x in the priority queue is $O(\log n)$ times the in-degree of x. The sum of in-degrees of all vertices is $O(m)$, and the bound follows. □

LEMMA 7. *Let p be the number of sources that appear during edge deletions on G. It takes $O(pm \log n)$ time to maintain the data structures for common-predecessor queries over all edge deletions, and they support queries in $O(p)$ time.*

PROOF. The above algorithm for a given source w is carried out for each of the $O(p)$ sources whenever an edge is deleted, in sorted order of sources. This allows us to label each vertex with a sorted list of sources that it has ceased to be a successor of as a result of the edge deletion. These can then be removed from its list of sources that it is reachable from in $O(p)$ time. A common-predecessor query takes $O(p)$ time to determine, for the two given vertices, whether the two sorted lists of sources that they are successors of contain a common element. □